VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMPONENT PRO SÉMANTICKÉ OBOHACENÍ

SEMANTIC ENRICHMENT COMPONENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAN DOLEŽAL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAROSLAV DYTRYCH, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/18884/2017/xdolez52

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Doležal Jan, Bc.**
Obor: Inteligentní systémy
Téma: **Komponent pro sémantické obohacení**
Semantic Enrichment Component
Kategorie: Zpracování řeči a přirozeného jazyka

Pokyny:

1. Seznamte se s prací se sdílenou pamětí v jazyce C a s jazyky Python a BASH.
2. Prostudujte nástroj pro rozpoznání pojmenovaných entit v textu (NER) Skupiny znalostních technologií (KNOT@FIT), jeho znalostní bázi (KB) a různé formáty pro uložení anotací.
3. Navrhněte program pro efektivní práci s KB ve sdílené paměti.
4. Navrhněte nástroj pro sémantické obohacení textu využívající sdílenou KB a daný NER, přičemž do budoucna bude možné integrovat i jiné nástroje NER. Nástroj umožní vícevláknové zpracování. Vstupem bude prostý či vertikální text a formát výstupu bude možné zvolit (např. NIF, XML, HTML ...). Službu bude možné využívat jak z příkazové řádky, tak i přes protokol HTTP.
5. Implementujte navržené řešení a odladťe pro servery KNOT a superpočítač Salomon.
6. Zhodnoťte dosažené výsledky a navrhněte možná rozšíření do budoucna.

Literatura:

- Dle doporučení vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dytrych Jaroslav, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato diplomová práce se zabývá komponentou pro sémantické obohacení textu (SEC), která ve vstupním textovém dokumentu nebo vertikálním textu vyhledá entity (např. osoby nebo místa) a informace o nich vrátí na výstup. Cíle této komponenty jsou vytvoření jednotného rozhraní pro nástroje rozpoznávající entity v textu, umožnění paralelního zpracování dokumentů, úspora operační paměti při využívání znalostní báze a zrychlení přístupu k jejímu obsahu. K tomu byl specifikován výstup pro nástroje rozpoznávající entity v textu, implementován nástroj pro uložení předzpracované znalostní báze do sdílené paměti a při tvorbě komponenty bylo využito schéma klient-server.

Abstract

This master's thesis describes Semantic Enrichment Component (SEC), that searches entities (e.g., persons or places) in the input text document and returns information about them. The goals of this component are to create a single interface for named entity recognition tools, to enable parallel document processing, to save memory while using the knowledge base, and to speed up access to its content. To achieve these goals, the output of the named entity recognition tools in the text was specified, the tool for storing the preprocessed knowledge base into the shared memory was implemented, and the client-server scheme was used to create the component.

Klíčová slova

sémantické obohacení textu, SEC, KNOT, rozpoznání jmených entit, NER, vyhledávání entit v KB, FIGA, znalostní báze, KB, anotace, anotace textu, Decipher, 4A, 4A systém, 4A anotační server, vertikální text, vertikál, porozumění textu, zpracování textu, význam textu, alternativní entity, SXML, paralelní zpracování, paralelizace, sdílení prostředků, sdílení zdrojů, sdílená znalostní báze, démon pro znalostní bázi, SharedKB, porovnávání rozpoznávacích nástrojů NER, porovnávání znalostních bází

Keywords

semantic enrichment of text, SEC, KNOT, named entity recognition, NER, searching of entities in KB, FIGA, knowledge base, KB, annotation, annotation of text, Decipher, 4A, 4A system, 4A annotation server, vertical text, vertical, understanding to the text, processing of the text, the meaning of the text, alternative entities, SXML, parallel processing, parallelization, resource sharing, sharing of the resources, shared knowledge base, deamon for knowledge base, SharedKB, comparison of NER tools, comparison of knowledge bases

Citace

DOLEŽAL, Jan. *Komponent pro sémantické obohacení*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Dytrych, Ph.D.

Komponent pro sémantické obohacení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Dytrycha. Další informace mi poskytli Ing. Lubomír Otrusina, doc. Pavel Smrž, Ing. Jan Kouřil a Dr. Petr Peringer. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Jan Doležal
30. května 2018

Poděkování

Děkuji svému vedoucímu diplomové práce Ing. Jaroslavu Dytrychovi za jeho čas, ochotu a cenné rady. Také děkuji za podporu a poskytnutý čas svým rodičům a sourozencům, díky čemuž jsem se mohl plně věnovat této práci.

Obsah

Kapitola 1

Úvod

Tato diplomová práce popisuje návrh a implementaci komponenty pro sémantické obohacení textu (SEC), která ve vstupním textovém dokumentu vyhledá entity a informace o nich vrátí na výstup. K tomu mimo jiné využívá nástroj pro vytvoření sdílené znalostní báze, navržený s důrazem na rychlost přístupu a minimální navýšení velikosti zdrojové znalostní báze.

Následující kapitola 2 rozšiřuje zadání práce. Definuje hlouběji její cíle a snaží přiblížit čtenáři jejich smysl.

Analýza současného stavu, uvedená v kapitole 3, popisuje blíže komponenty, které jsou využity v návrhu.

Kapitola 4 popisuje technologie, jenž budou využity při implementaci. Obsahuje teorii nutnou k vytvoření programu pro efektivní práci se znalostní bází ve sdílené paměti. Dále obsahuje popis formátů pro uložení anotací, jenž jsou vyžadovány i jako výstupní formáty komponenty pro sémantické obohacení textu.

Návrh řešení se nalézá v kapitole 5. Zde se čtenář dozví o možných překážkách při implementaci a jejich řešení. Kromě návrhu programu pro efektivní práci ve sdílené paměti a komponenty pro sémantické obohacení je zde například specifikována hlavička znalostní báze, výstupní formát SXML komponenty pro sémantické obohacení textu s návrhy vhodných entit k názvům, jenž je zastupují a specifikaci výstupu nástroje NER.

Implementace výsledného řešení dle návrhu je pak popsána v kapitole 6. Je zde popsáno rozdělení komponenty pro sémantické obohacení textu do tří spustitelných souborů a jejich funkce. Dále je popsána funkce nástrojů pro sdílení znalostní báze, tedy démona pro sdílení znalostní báze a knihovny pro přístup k ní.

Po implementaci byla komponenta důkladně otestována v prostředí serverů Výzkumné skupiny znalostních technologií FIT VUT v Brně, na superpočítači Salomon i v rámci praktického nasazení v jiných projektech. Testování a nasazení je popsáno v kapitole 7.

Závěrečná kapitola 8 shrnuje a hodnotí výsledky této diplomové práce a zaměřuje se na to, jak bude práce pokračovat.

Kapitola 2

Cíle práce

Hlavním cílem je navrhnout komponentu pro sémantické obohacení textu (SEC – Semantic Enrichment Component). SEC má využívat nástroj NER (Named Entity Recognition – rozpoznávání pojmenovaných entit v textu), nástroj Figa a znalostní bázi Výzkumné skupiny znalostních technologií (KNOT@FIT). Musí umožnit vícevláknové zpracování a poskytovat služby přes příkazovou řádku a protokol HTTP.

Hlavní službou bude anotace textu, k čemuž má být využit nástroj NER a znalostní báze. Jejím vstupem bude prostý či vertikální text v kódování UTF-8, zabalený společně s názvem a atributy služby do formátu JSON. Konfiguraci (název a atributy služby) bude možné uvést zvlášť, aby bylo možné na vstup zadat prostý či vertikální text. Pro anotaci textu bude možné využít i jiné nástroje NER. To umožní porovnat různé nástroje NER skrze jednotné rozhraní (např. v jejich úspěšnosti, rychlosti apod.).

Služba pro anotaci textu SEC bude umožňovat výstupy do různých formátů dle vstupního formátu. Pro dokument v prostém textu umožní výstup do formátů HTML, XML, RDF, NIF, Text, Index a Index 2. Je třeba navrhnout specializovaný výstupní formát Suggestion XML (v této práci zkráceně jako SXML) s návrhy vhodných entit k názvům, jenž je zastupují, pro vnitřní použití ve skupině KNOT@FIT. SXML bude aplikací univerzálního značkovacího metajazyka XML a bude obsahovat pouze anotovaný text. Především bude tento výstupní formát využívat anotační server 4A skupiny KNOT@FIT. Pro dokumenty ve vertikálním textu SEC umožní výstup do formátů MG4J, Manatee, Manatee 2 a ElasticSearch.

Dále bude poskytovat službu vyhledání entity dle jejího názvu, popř. počátku názvu, k čemuž využije nástroj Figa. Třetí služba umožní vypsat typy entit ve znalostní bázi a jejich atributy. Uvedené služby a jejich konfigurace přes formát JSON vychází z dokumentu [1] pro evropský projekt Decipher.

K efektivní práci se znalostní bází SEC využije program v jazyce C, navržený v této diplomové práci, pracovně nazvaný „Démon pro znalostní bázi“. Smyslem démona bude odlehčit operační paměti při použití znalostní báze více nástroji na jednom stroji. Aby se znalostní bází mohl každý nástroj rychle pracovat a nečekat na disk, načítá si ji celou do operační paměti. Operační paměť pak rychle dochází ze dvou důvodů:

- Znalostní báze je načtena ve formátu Unicode a všechny znaky mají v paměti stejnou velikost, větší než `char` (např. v jazyce Python verze 2.x je Unicode ukládán do operační paměti v kódování UCS-2 nebo UCS-4, tedy 16 nebo 32 bitů na znak[1]).

[1] Unicode v jazyce Python verze 2.x – https://docs.python.org/2/c-api/unicode.html

- Procesy znalostní bázi nemění, přesto ji každý proces má načtenou jen pro sebe (nesdílí ji).

Ze sdílené paměti si budou moci jiné nástroje přebírat data, která potřebují, a nemusí tak plýtvat operační pamětí. K výslednému programu má být vytvořena knihovna a demonstrační program, který přes knihovnu bude získávat informace ze znalostní báze ve sdílené paměti.

Veškeré nástroje využívající znalostní bázi pouze pro čtení budou následně upraveny, aby využívaly démona pro znalostní bázi. Tímto způsobem se centralizuje popis atributů entit obsažených ve znalostní bázi. Nástroje mají popis znalostní báze často uveden přímo v kódu a změna znalostní báze (např. přidání typu entity, přidání atributu typu apod.) pak vyžaduje změnu popisu ve všech nástrojích, kterých se to týká.

Aby mohl SEC pracovat ve více vláknech je důležitá především úprava nástroje NER. Ten totiž vícevláknové zpracování neposkytuje a jeho běh vyžaduje řádově jednotky gibibajtů paměti. Využití démona pro znalostní bázi nástrojem NER ušetří operační paměť a NER pak bude možné spouštět ve více instancích. Jeho úprava tak umožní dokumenty zpracovávat paralelně nástrojem SEC. Dále bude vhodné zrychlit inicializaci nástroje NER a vylepšit jeho rozlišovací schopnost vytvořením nástroje pro hledání dat, jenž každé datum ve vstupním textu (psaném anglicky) popíše převedením do formátu ISO 8601. Ing. Otrusina mi k tomu doporučil použít jediný komplexní regulární výraz.

Veškerá implementace musí být odladěna pro servery KNOT a superpočítač Salomon. Cílovým operačním systémem je GNU/Linux, architektura pak x86_64. Pro superpočítač Salomon [2] je třeba vytvořit skripty v jazyce BASH pro spuštění SEC na výpočetních uzlech a distribuci úloh mezi nimi [3].

Možným rozšířením SEC bude *testovací mód*, jenž umožní testovat práci se strukturovanými anotacemi, které nástroj NER neumí vytvářet. Testovací mód rozšiřuje službu anotace textu. V jejím atributu „document_uri“ vyhledá klíč „tid“ a dle jeho hodnoty a požadovaného výstupního formátu pak v adresáři, zadaném atributem při spuštění nástroje SEC, otevře odpovídající soubor. Pokud je takový soubor nalezen, pak místo výsledků z nástroje NER bude odpovědí na zadaný dotaz obsah toho souboru. Kromě klíče „tid“ může mít atribut „document_uri“ klíč „aid“. Je-li dle jeho hodnoty nalezen soubor, pak jeho obsah pouze obohatí výsledek nástroje NER (připojí se k němu). Vzhledem k tomu, že se strukturovanými anotacemi v současné době pracuje pouze anotační nástroj 4A (viz dále), bude postačující, když testovací mód bude fungovat pouze pro specializovaný výstupní formát SXML.

Kapitola 3

Analýza současného stavu

V této kapitole jsou popsány komponenty, které (úzce) souvisí s cílem práce z kapitoly 2. Uvedené komponenty byly vytvořeny Výzkumnou skupinou znalostních technologií naší fakulty (KNOT@FIT). Základní komponentou je znalostní báze (uvedená v sekci 3.1), následuje nástroj Figa, který umožňuje efektivně ve znalostní bázi vyhledávat (sekce 3.2), a nástroj NER, jenž předchozí dvě komponenty využívá (sekce 3.3).

3.1 Znalostní báze

Znalostní báze (KB) nástroje pro rozpoznání pojmenovaných entit v textu (NER) je textový soubor v kódování UTF-8 ve formátu TSV (tabulátory oddělené hodnoty). Jejími položkami jsou entity (více či méně známé osoby, místa, výtvarná díla apod.) a jejich atributy (např. u osoby: jméno, příjmení, datum/místo narození/popř. úmrtí apod.). Aktuálně obsahuje přes 5 milionů řádků a její velikost přesahuje 2 GiB[1]. Na každém řádku je uložena jedna entita, jejíž typ je uveden v druhém atributu (sloupci). Typy entit se liší počtem atributů (sloupců oddělených tabulátory) a jejich významem [4]. Některé atributy mohou obsahovat více hodnot oddělených znakem „|“.

Aktuálně má znalostní báze 14 typů entit, ale lze počítat s tím, že se jejich počet ještě zvýší. Dále se počítá s tím, že se časem u některých typů zavedou podtypy, které budou rozšiřovat stávající typy. Entita určitého typu bude moci mít více podtypů, čímž se rozšíří počet atributů (např. nějaká entita osoba může být malířem i hudebníkem, o malíři bychom rádi věděli, jaké obrazy namaloval, o hudebníkovi jakými skladbami se proslavil).

Znalostní báze se liší podle účelu použití. Sama o sobě nemá veliký význam. Ten získá, až když ji některý nástroj začne využívat, popř. když vznikne právě pro nějaký nástroj. V tomto případě jí smysl dává až koncový uživatel. Obsažené znalosti ve znalostní bázi byly původně zaměřeny na kulturní dědictví, dnes jsou zaměřeny obecně, i když původní informace v ní převážně zůstaly.

Entity i atributy byly získány z volně dostupných zdrojů (jako např. Wikipedia[2], Freebase[3], DBpedia[4] apod.). Pomocí několika nástrojů byly entity jednotlivých typů i s atributy extrahovány do dílčích znalostních bází. Pro každý typ byly sekvenčně sloučeny zdroje, přičemž byly detekovány duplikáty, které byly odstraněny sloučením atributů. Nakonec byly sloučeny výsledné dílčí znalostní báze jednotlivých typů do výsledné znalostní báze.

[1] Znalostní bázi je možné stáhnout na adrese: http://athena3.fit.vutbr.cz/kb/KB.all
[2] https://dumps.wikimedia.org/
[3] https://developers.google.com/freebase/
[4] http://wiki.dbpedia.org/

3.2 Nástroj Figa

Aby bylo možné se znalostní bází efektivně pracovat, je třeba ji účelově předzpracovat, tedy urychlit ta vyhledávání, která jsou v ní nejčastější. Pro tuto činnost byl výzkumnou skupinou KNOT@FIT vytvořen nástroj Figa (FIt GAzetteer).

Pomocným skriptem se tomuto nástroji předají dvojice (hledaný výraz, číslo řádku do znalostní báze), čímž si vytvoří datové soubory. Poté tyto datové soubory slouží k vyhledávání entit ve znalostní bázi. Každý datový soubor je zaměřen na jiný atribut entity (např. na název nebo na URI) [5].

Nástroj Figa byl dříve založen na konečných automatech doktora Jana Daciuka z polské Politechniki Gdańskiej[5]. Nyní používá slovníky CEDAR nebo DARTS [6]. Je implementován v jazyce C++ a obalen pomocí nastroje SWIG pro jazyk Python.

Použití nástroje Figa

Nástroj Figa má dvě možnosti, jak může být použit. Následující informace vychází z webové stránky skupiny KNOT k nástroji Figa[6]. První možností je využít jej k vyhledání jmen v textu, dle jmen obsažených ve slovníku (datovém souboru nástroje Figa). Tuto možnost využívá ke své funkci nástroj NER a bude využita i pro SEC. Na vstupu takto očekává textový dokument a na standardní výstup vrací nalezené entity. Pro úplnost zde popíši syntaxi výstupního formátu v Backusově-Naurově formě (BNF):

```
<výstup Figa> :== <řádek výstupu>
    | <řádek výstupu> <výstup Figa>

<řádek výstupu> :== <čísla řádků do KB> "\t" <počáteční offset>
      "\t" <koncový offset> "\t" <fragment> "\t" <příznak> "\n"

<čísla řádků do KB> :== <číslo>
    | <číslo> ";" <čísla řádků do KB>
```

kde:

- `<čísla řádků do KB>` odkazují na řádky ve znalostní bázi s entitami, jenž mají mezi atributy `<fragment>`,
- `<počáteční offset>` a `<koncový offset>` jsou pozice prvního a posledního znaku řetězce `<fragment>` (na pozici 1 leží první znak vstupního textu),
- `<příznak>` může nabývat dvou hodnot: „F“ – fragment plně odpovídá atributu odkazovaných entit; „S“ – byl tolerován překlep ve fragmentu.

Druhou možností, nazvanou „Autocomplete“, je využít nástroj Figa k vyhledávání entit, u kterých prefix jejich názvu odpovídá zadanému řetězci. Tato možnost bude využita pro SEC. Výstupní formát je následující (v BNF):

```
<výstup Autocomplete> :== <řádek výstupu>
    | <řádek výstupu> <výstup Autocomplete>
```

[5] Jan Daciuk — viz http://www.jandaciuk.pl/ a https://pg.edu.pl/jandac
[6] Nástroj Figa — viz http://knot.fit.vutbr.cz/Decipher_NER/decipher_fsa_cs.html

```
<řádek výstupu> :== <název entity> "\t" <čísla řádků do KB> "\n"

<čísla řádků do KB> :== <číslo>
    | <číslo> ";" <čísla řádků do KB>
```

kde:

- `<název entity>` je řetězec, jehož prefix odpovídá zadanému řetězci,
- `<čísla řádků do KB>` odkazují na řádky ve znalostní bázi s entitami, jejichž název se shoduje s `<název entity>`.

3.3 Nástroj NER

Nástroj pro rozpoznání pojmenovaných entit v textu (NER) byl vytvořen výzkumnou skupinou KNOT@FIT v rámci evropského projektu Decipher, zaměřeného na kulturní dědictví. Více o tomto projektu je možné nalézt v diplomové práci Ing. Miloše Cundráka [7][7]. Později byl nástroj NER zobecněn a použit jako komponenta systému 4A[8] skupiny KNOT@FIT. Je implementován v jazyce Python 2.7.*x*.

NER vyhledává v dokumentech názvy zastupující entity ve znalostní bázi (popsané v kapitole 3.1). U každého nalezeného názvu poté určuje, která entita mu nejlépe odpovídá dle názvu samotného, statistiky uložené ve znalostní bázi a následně dle kontextu. Předpokládá se, že vstupní dokumenty jsou prostým textem v kódování UTF-8 a strukturovány pouze do vět (souvětí) a odstavců tak, jak je to u lidsky čitelného textu běžné [8].

Nástroj NER umí pracovat ve dvou režimech. Výchozí režim očekává dokument v souboru specifikovaném atributem nebo na standardním vstupu. Režim „daemon“ umožňuje na jedno spuštění zpracovat dokumentů více [9]. Dokumenty postupně čte a zpracovává ze standardního vstupu programu, oddělené řetězcem `NER_NEW_FILE`. Výstupem nástroje NER jsou zvlášť na řádcích nalezené názvy s jejich polohou v dokumentu, typem a popř. řádkem ve znalostní bázi s nejlépe odpovídající entitou.

[7] Původní stránka projektu byla http://decipher-research.eu/, nyní bohužel neobsahuje relevantní informace.

[8] Systém 4A — viz http://knot.fit.vutbr.cz/annotations/

Kapitola 4

Využité technologie

Tato kapitola předává čtenáři informace k základním stavebním kamenům použitým v návrhu programu. Kapitola je také přizpůsobena tomu, že program má pracovat na Linuxových operačních systémech. Nemá tedy za cíl být encyklopedickým přehledem, pouze uvádí možnosti, se kterými jsem se musel seznámit.

4.1 Práce se sdílenou pamětí

Jedním z prostředků pro meziprocesovou komunikaci (IPC) je sdílená paměť. Je to oblast v operační paměti sdílená operačním systémem mezi dvěma a více procesy. Jedná se o nejrychlejší metodu IPC, jelikož nevyžaduje žádné zásahy jádra operačního systému během samotné komunikace [10, s. 921]. Změnu hodnoty ve sdílené paměti jedním procesem mohou ihned reflektovat ostatní procesy [10, s. 37].

Dalším z důvodů, proč sdílet paměť mezi více procesy, je úspora operační paměti (např. u rozsáhlých databází). Vždy je však nutné, aby zápis byl prováděn obezřetně, aby byla zachována konzistence dat, tedy s výlučným přístupem čtenářů a písařů. Pokud jeden proces zapisuje do nějakého logického bloku, nemůže už jiný proces zapisovat ani číst do/z tohoto bloku, jinak by mohlo dojít k nekonzistenci dat. Nezapisuje-li žádný z procesů do sdíleného místa v paměti, pak mohou procesy z tohoto místa libovolně číst. K předejití konfliktu mezi zapisujícími procesy je vhodné využít některé synchronizační metody (jako např. semafor či signál).

Na POSIXových systémech existují tři API (rozhraní pro programování aplikací), zpřístupňující systémová volání pro správu sdílené paměti. Starším API je *System V shared memory*, novějším pak *POSIX shared memory*, které poskytuje jednodušší rozhraní [11], a *Memory mappings*. Pro implementaci jsem si vybral *POSIX shared memory* a s ní související *Memory mappings*, které nyní popíši. Není-li uvedeno jinak, informace uvedené v této sekci byly čerpány z [10].

Memory mappings

Systémové volání `mmap()` umožňuje vytvořit nové mapování paměti ve virtuální paměti volajícího procesu. Umožňuje mapovat do paměti soubor nebo alokovat blok operační paměti (tzv. anonymní mapování). Změny mapované paměti je poté možné sdílet mezi procesy a využít tak k meziprocesové komunikaci (IPC). Funkce `mmap()` poskytuje čtyři odlišné typy mapování paměti:

- *Soukromé mapování souboru*: Obsah mapované paměti je přímo mapovaný soubor v režimu *copy-on-write* (COW). Všechny procesy mapující stejný soubor tímto způsobem mají zpočátku mapované stejné fyzické stránky paměti. Pokusí-li se však nějaký proces o zápis, dojde k vytvoření kopie upravované stránky a v ní dojde ke změně. Provedené změny tedy nejsou viditelné pro ostatní procesy, ani se neprojeví na mapovaném souboru.

- *Soukromé anonymní mapování*: Procesu je takto přidělena volná část operační paměti, jejíž obsah je inicializován nulou. Je to tedy podobné soukromému mapování souboru `/dev/zero`. Pokud si proces systémovým voláním `fork()` vytvoří potomka, pak potomek dědí takto mapovanou paměť v režimu COW. Rodičovský proces a potomek si navzájem nevidí na změny, které udělali. Funkce `malloc()` tento typ mapování systémovým voláním `mmap()` používá pro alokaci velkých bloků paměti.

- *Sdílené mapování souboru*: Umožňuje procesům do paměti mapovat stejnou oblast souboru, dělat změny v jeho obsahu a tímto způsobem si mezi sebou předávat informace. Jedná se o alternativu k funkcím `read()` a `write()` nad otevřeným souborem. Z hlediska výkonu však může být v některých případech rychlejší, protože šetří režie se systémovým voláním. Eliminuje kopírování mezi rychlou vyrovnávací pamětí jádra a vyrovnávací pamětí v uživatelském prostoru, čímž i snižuje nároky na paměť.

- *Sdílené anonymní mapování*: Podobně jako u soukromého anonymního mapování je procesu přidělena volná část operační paměti s obsahem inicializovaným nulou. Rozdíl je v tom, že potomek, vytvořený systémovým voláním `fork()`, má možnost číst i zapisovat z/do fyzicky stejné paměti (takto mapované) jako rodičovský proces a tímto způsobem tak mohou mezi sebou komunikovat.

Odmapování namapované paměti umožňuje systémové volání `munmap()`.

POSIX shared memory

Toto API vzniklo za účelem odstranění potenciálních nedostatků předchozích dvou technik. System V shared memory nepoužívá standardní UNIXový model, tedy místo souborů používá IPC objekty, ke kterým je možné se připojit pomocí unikátního celočíselného klíče. Použití mapování paměti zase vyžaduje vytvoření souboru na disku, aby mohla být paměť sdílena mezi procesy, jenž si odkaz na ni nemohly předat systémovým voláním `fork()`. To s sebou nese režii pomalejší paměti (typicky sekundární paměti).

Každý mechanismus POSIX IPC umožňuje vytvořit nebo otevřít objekt voláním končícím na *_open*. Objekt je fyzicky přístupný přes souborový systém a klíčem k němu je název souboru. Je možné mu nastavit přístupová pravá podobně jako u běženého se souboru, a to včetně práv ACL[1]. Typicky jsou tyto soubory uloženy v adresáři `/dev/shm/`. Které konkrétní funkce mechanismus POSIX shared memory nabízí, se lze dočíst v [11]. Mimo ně využívá standardní systémová volání pro práci s popisovači souborů.

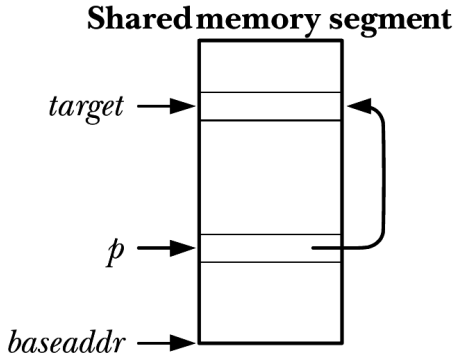
Ukládání ukazatelů ve sdílené paměti

Chceme-li mít ve sdílené paměti ukazatel na jiný objekt (v téže sdílené paměti), pak máme dvě možnosti, jak to realizovat:

[1] ACL (seznam řízení přístupu) je v tomto kontextu seznam dalších přístupových práv pro konkrétní uživatele a skupiny, připojený k objektu. Je podporován od Linuxového jádra verze 2.6.19. [11]

- Sdílenou paměť namapujeme vždy na stejné místo. Toto ovšem není doporučeno z následujících důvodů:
  - Redukuje to přenositelnost aplikace. Adresa validní v jedné implementaci UNIXu může být nevalidní v jiné.
  - Tato adresa ve virtuální paměti procesu již nemusí být volná. Například na ni již byla namapována paměť voláním knihovní funkce, namapován jiný segment sdílené paměti nebo sdílená knihovna.
- Adresa ukazatele v paměti bude relativní od počátku segmentu sdílené paměti. To vyžaduje výpočet absolutní adresy v rámci virtuální paměti procesu součtem (relativní) adresy uložené v ukazateli a adresy počátku segmentu sdílené paměti.

Pokud chceme dodržet doporučenou praktiku, tedy nechat výběr adresy pro segment sdílené paměti na jádře, ukazatel ve sdílené paměti nemůže nést absolutní adresu v rámci virtuální paměti procesu. Řešením je uložit do tohoto ukazatele relativní adresu od počátku segmentu sdílené paměti. Je to z toho důvodu, že každý proces může mít sdílenou paměť namapovanou na jinou adresu ve virtuální paměti.



Obrázek 4.1: Použití ukazatele v segmentu sdílené paměti (převzato z [10])

Vezměme jako příklad obrázek 4.1. Mějme adresu počátku segmentu sdílené paměti `baseaddr`, ve sdílené paměti objekt na adrese `target` a ukazatel na adrese `p`. Může to být realizováno například tak, že na začátku sdílené paměti bude struktura obsahující ukazatel `p`. Nebo se může jednat o pole polí, tedy pole ukazatelů na pole. Pokud do ukazatele `*p` uložíme adresu `target`, pak v jiném procesu může být adresa v ukazateli `*p` různá od adresy `target`.

```
*p = target; /* Chybné uložení adresy target do ukazatele *p. */
```

Chceme-li, aby ukazatel `*p` ukazoval na objekt, pak musí nést adresu relativní (tedy ofset).

```
*p = (target - baseaddr); /* Uložení ofsetu do *p. */
```

Součtem (relativní) adresy uložené v ukazateli `*p` a adresy počátku segmentu sdílené paměti `baseaddr` získáme adresu `target` (objektu ve sdílené paměti, na který `*p` ukazuje).

```
target = baseaddr + *p; /* Použití ofsetu pro přístup k target. */
```

4.2 Jazyk Python

Python je interpretovaný, interaktivní, vysokoúrovňový a objektově orientovaný programovací jazyk, navržený v roce 1991 Guido van Rossumem [12]. Zahrnuje moduly a výjimky, je dynamicky typovaný a využívá prvky z funkcionálního programování. Jazyk je vyvíjen nadací Python Software Foundation pod licencí kompatibilní s GNU GPL [13]. Aktuálně jsou používány dvě nekompatibilní verze 2.x a 3.x, z nichž nejnovější jsou verze 2.7.14 a 3.6.4 [14]. Vývojáři již od roku 2008 plánovali ukončení podpory pythonu 2.x na rok 2015, následně však tento termín posunuli na rok 2020 [15]. Další informace o tomto jazyce je možné nalézt v knize Allena B. Downeyho *How to Think Like a Computer Scientist: Learning with Python* [16][2].

4.3 Jazyk BASH

BASH (Bourne-Again SHell) je interpret a programovací jazyk umožňující jednoduše vykonávat příkazy a propojovat nástroje GNU [17]. Poprvé byl vydán v roce 1989 Brianem Foxem. Nyní jej spravuje nadace Free Software Foundation a je vydán pod licencí GNU GPL. Aktuální verze je 4.4.12 [18]. Další informace o tomto jazyce je možné nalézt v knize *LINUX v kostce* [19].

4.4 HTTP

HTTP (HyperText Transfer Protocol) je protokol aplikační vrstvy modelu ISO/OSI pro výměnu dat typu dotaz-odpověď. Je používán pro web od roku 1990. Současně používanou verzí je HTTP/1.1 definovaná v RFC 2616 [20] v roce 1999. Nejnovější verzí je pak HTTP/2 definovaná v RFC 7540 [21] v květnu roku 2015. Protokol HTTP je dle zadání využit pro komunikaci klient-server s komponentou pro sémantické obohacení textu (SEC).

4.5 JSON

JSON[3] (JavaScript Object Notation) je formát pro výměnu dat, jenž je podmnožinou jazyka JavaScript. Byl vymyšlen Douglasem Crockfordem[4] na počátku roku 2000 a jeho nejnovějším standardem je RFC 8259 [22] z prosince roku 2017. Tento formát je využit pro vstup i výstup komponenty pro sémantické obohacení textu (SEC), především pro konfiguraci zpracování vstupu.

4.6 Vertikální text

Vertikální text (nebo také vertikál) je textový soubor, jenž na každém řádku obsahuje buď řídící značku nebo slovo či nějaký znak (interpunkce apod.). Dle řídících značek se slova skládají do vět, odstavců či celých dokumentů. Řídící značky jsou podobné formátu XML, ale celý dokument formátu XML neodpovídá, protože neobsahuje kořenovou značku. Více

[2] Jazyk Python — Překlad knihy *How to Think Like a Computer Scientist: Learning with Python* je volně dostupný pod názvem *Učíme se programovat v jazyce Python 3* na http://howto.py.cz (přeložil Jaroslav Kubias).

[3] JSON — https://json.org/json-cz.html

[4] JSON — https://www.tbray.org/ongoing/When/201x/2014/03/05/RFC7159-JSON

o vertikálním formátu používaném výzkumnou skupinou KNOT@FIT lze nalézt v bakalářské práci Miloše Šváni [23], popř. na webových stránkách [24].

4.7 Formáty pro uložení anotací

Anotace lze ukládat do celé řady formátů. Ve Výzkumné skupině znalostních technologií (KNOT) se využívají níže uvedené formáty, které jsou vyžadovány i jako výstupní formáty SEC.

Formáty HTML a Text jsou určené pro rychlé kontroly výstupů nástrojů NER lidmi. Ve formátu HTML se uživatel zaměří převážně na anotovaný text a správnost anotací v jeho kontextu. Ve formátu Text lze rychle kontrolovat atributy jednotlivých anotací, tedy správnost dat získaných ze znalostní báze.

Formáty RDF, XML a NIF jsou vhodné pro strojové zpracování. RDF (a jeho alternativní formát XML) je zaměřen především na automatizované odvozování znalostí z anotací běžně využívanými nástroji, zatímco NIF je zaměřen na jednoduchou práci se samotnými anotacemi. Formát NIF byl využit např. v práci Bc. Dávida Prexty [25], která se zabývá vytvořením univerzálního nástroje pro srovnávání různých nástrojů NER (viz níže).

Formáty Index, Index 2, MG4J, Manatee a Manatee 2 jsou určeny k přípravě anotovaných dat ve vertikálním formátu pro indexaci a následné vyhledávání nad indexy rozsáhlých textových dat. Nejstaršími formáty jsou Index a Index 2, které byly ve výzkumné skupině KNOT využívány pro indexaci nástrojem ElasticSearch (druhý z formátů je uzpůsoben pro využití speciálního zásuvného modulu). Později výzkumná skupina začala pracovat s korpusy velkých textových dat z webu z projektu Common Crawl[5] a ukázalo se, že výkonnostní limity nástroje ElasticSearch neumožňují indexaci v přijatelném čase. Proto přešla na vlastní indexační nástroj postavený na MG4J[6], který vyžaduje specializovaný formát anotací. Protože však ani toto řešení nedosahuje požadovaných výsledků (zejména kvůli problémům při práci s méně častými znaky v textu, nemožnosti aktualizace indexu za běhu apod.), bylo dále experimentováno se systémem NoSketch Engine[7], resp. s jeho nástrojem Manatee. Za účelem realizace těchto experimentů byly vytvořeny formáty Manatee a Manatee 2.

HTML

HTML (HyperText Markup Language) je značkovací jazyk používaný k tvorbě webových stránek. Je aplikací univerzálního značkovacího metajazyka SGML. O vývoj se stará konsorcium W3C (World Wide Web Consortium) přibližně od roku 1995. Poslední vydanou verzí je HTML 5.2 [26]. Další informace o tomto jazyce je možné nalézt v knize [27].

Vzor pro výstup nástroje pro sémantické obohacení textu

Výstupní formát HTML nástroje pro sémantické obohacení textu (SEC) byl dle zadání vytvořen dle vzoru vytvořeného skupinou KNOT@FIT[8]. Převzat byl především způsob zobrazení anotací (vyskakovací okno) v HTML a jejich struktura, která je využita pro barevné odlišení typů anotací. Syntax anotace v Backusově-Naurově formě (BNF) je následující:

[5] http://commoncrawl.org/

[6] http://mg4j.di.unimi.it/

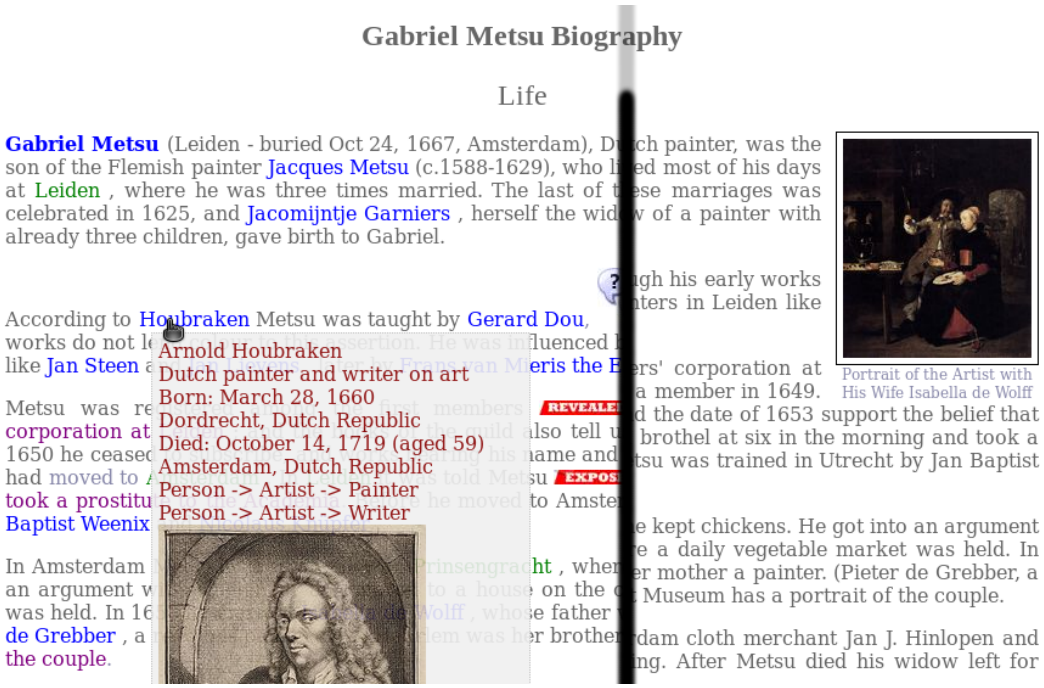
[7] https://nlp.fi.muni.cz/trac/noske

[8] Vzor anotací v HTML lze nalézt na adrese http://knot.fit.vutbr.cz/decipher/pomocne/Gabriel%20Metsu%20Biography.html a původní stránku bez anotací na adrese https://web.archive.org/web/20130707155948/http://www.gabrielmetsu.org:80/biography.html

```
<anotace v HTML> :== ’<span class="annotation">’
      <anotovaný text> <blok anotací k textu> ’</span>’

<anotovaný text> :== ’<span class="annotation ’ <typ_anotace> ’">’ <text> ’</span>’
<typ_anotace> :== <typ entity> | ’coreference_’ <typ entity>

<blok anotací k textu> :== ’<span class="annotation_features">’
      <text typu anotace> <mapa k anotaci location> <atributy> ’</span>’

<text typu anotace> :== ’<span class="annotation_type">’ <typ anotace>
      ’</span><br>’
<typ anotace> :== <typ entity> | ’Coreference to ’ <typ entity>

<atributy> :== <atribut> | <atribut> <atributy>
<atribut> :== ’<span class="att_name ’ <název atributu> ’">’ <název atributu>
      ’: </span><span class="att_value ’ <název atributu> ’">’ <hodnota atributu>
      ’</span><br>’
```

kde:

- `<typ_anotace>` určuje typ entity, popř. zda se jedná o odkaz na dříve uvedenou entitu,
- `<mapa k anotaci location>` je v případě, že je entita typu `location` a obsahuje ve znalostní bázi zeměpisné souřadnice, nahrazen za prvek HTML `<iframe>` s Mapami Google na těchto souřadnicích,
- `<hodnota atributu>` může být obrázek, odkaz či text.



Obrázek 4.2: Ukázky vzoru s anotací a bez anotace ze stránek odkazovaných pod čarou[8].

RDF

RDF (Resource Description Framework) je struktura vyjadřující informace o zdrojích (dokumentech, lidech, fyzických objektech, ...) [28]. Datový model RDF obsahuje výroky o zdrojích. Formátem výroku je trojice subjekt, predikát a objekt. Subjekt a objekt reprezentují dva zdroje a predikát reprezentuje vztah orientovaný od subjektu k objektu (vlastnost) [28]. Tvoří tak ohodnocený orientovaný graf, kde uzly reprezentují zdroje a hrany vlastnosti dle hodnoty.

Pro zápis grafu RDF existuje několik formátů, z nichž některé jsou popsány taktéž v základní učebnici RDF [28]. Graf může být zapsán například do formátu RDF/XML či RDFa. Pro jazyk Python existuje knihovna RDFLib[9], jenž umožňuje vytvářet grafy RDF a serializovat např. do zmíněného formátu RDF/XML.

Struktura RDF je vyvíjena pod záštitou konsorcia W3C (World Wide Web Consortium). Poslední vydanou verzí je RDF 1.1 [28]. Příklad anotace v tomto formátu je následující:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:a="http://knot.fit.vutbr.cz/annotations/AnnotSchema/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <a:location rdf:nodeID="Nbadead3876c147feaf0b63bcd14f0eb9">
    <a:name>London</a:name>
  </a:location>
  <a:artist rdf:nodeID="N68c5e429561e4d06a2613bc08e7a382d">
    <a:display_term>Sir William Orpen</a:display_term>
  </a:artist>
</rdf:RDF>
```

XML

XML (eXtensible Markup Language) je univerzální značkovací metajazyk, jenž vznikl omezením jazyka SGML. Byl vyvinut pod záštitou konsorcia W3C (World Wide Web Consortium) skupinou XML Working Group v roce 1996. Je standardizován a nadále udržován konzorciem W3C. Poslední verzí je XML 1.1 [29]. O tomto jazyce je možné se dozvědět více z knihy [30].

Pro ukládání anotací ve formátu XML se využívá RDFa[10] – tedy vlastnosti entit jsou uloženy v atributech elementů XML, kterými jsou obaleny textové reprezentace jednotlivých entit. Ve formátu XML je uložen kompletní anotovaný text. Příklad anotovaného textu v tomto formátu je následující:

```
<annotation><artist display_term="Sir William Orpen" wikipedia_url="">William
    Orpen</artist> was an Irish portrait painter, who worked mainly in <location
    name="London">London</location></annotation>
```

NIF

Formát NIF[11] je založený na formátu RDF a umožňuje výměnu dat mezi nástroji pro zpracování přirozeného jazyka. Příklad anotace v tomto formátu je následující:

[9] RDF — https://www.w3.org/2001/sw/wiki/RDFLib a https://rdflib.readthedocs.io/en/stable/gettingstarted.html

[10] Resource Description Framework in Attributes https://www.w3.org/TR/rdfa-primer/

[11] http://persistence.uni-leipzig.org/nlp2rdf/

```
<http://wiki-link.nlp2rdf.org/data/1a/0a/06434d8769660b5aabe0e047ccb5/67
    cd7b68aa9a54ab8db92562b2a6edca#char=0,13>
       a~nif:String , nif:RFC5147String ;
       nif:referenceContext <http://wiki-link.nlp2rdf.org/data/1a/0a/06434
           d8769660b5aabe0e047ccb5/67cd7b68aa9a54ab8db92562b2a6edca#char=0,> ;
       nif:anchorOf "William Orpen"^^xsd:string ;
       nif:beginIndex "0"^^xsd:long ;
       nif:endIndex "13"^^xsd:long ;
       a~nif:Phrase ;
       itsrdf:taIdentRef <http://collection.britishcouncil.org/artist/artist
           /30783/17590> .

<http://wiki-link.nlp2rdf.org/data/1a/0a/06434d8769660b5aabe0e047ccb5/67
    cd7b68aa9a54ab8db92562b2a6edca#char=66,72>
       a~nif:String , nif:RFC5147String ;
       nif:referenceContext <http://wiki-link.nlp2rdf.org/data/1a/0a/06434
           d8769660b5aabe0e047ccb5/67cd7b68aa9a54ab8db92562b2a6edca#char=0,> ;
       nif:anchorOf "London"^^xsd:string ;
       nif:beginIndex "66"^^xsd:long ;
       nif:endIndex "72"^^xsd:long ;
       a~nif:Phrase ;
       itsrdf:taIdentRef <http://en.wikipedia.org/wiki/London> .
```

Text

Formát Text je jednoduchý formát čitelný člověkem. Obsahuje pouze anotace bez vstupního textu. Příklad anotace v tomto formátu je následující:

```
William Orpen
=============
[artist]
display_term: Sir William Orpen

London
======
[location]
name: London
```

Index

Index je formát určený pro indexaci nástrojem ElasticSearch[12], jenž se používá jako fulltextový vyhledávač. Příklad anotace v tomto formátu je následující:

```
William Orpen[artist;Sir_William_Orpen_displayterm] was an Irish portrait painter,
     who worked mainly in London[location;London_name].
```

Index 2

Tento formát je obdobou předchozího formátu Index. Byly provedeny drobné změny pro využití speciálního zásuvného modulu k fulltextovému vyhledávači ElasticSearch. Příklad anotace v tomto formátu je následující:

[12] https://www.elastic.co/products/elasticsearch

```
William Orpen[displayterm_Sir_William_Orpen] was an Irish portrait painter, who
    worked mainly in London[name_London].
```

MG4J

Formát MG4J je určený pro indexaci nástrojem Výzkumné skupiny znalostních technologií (KNOT) vytvořeným v rámci projektu Stahování, zpracování a indexování rozsáhlých textových korpusů[13], který je postavený na nástroji MG4J[14]. Jedná se o 27 sloupcový vertikální formát navržený danou výzkumnou skupinou. První až 13. sloupec obsahují samotný text ve vertikálním formátu a syntaktické informace. Následují sloupce obsahující sémantické informace, tedy anotace. Významy sémantických sloupců `param0` – `param9` jsou dané typem anotace v 15. sloupci.

Příklad anotace v tomto formátu je následující:

```
1 FILENAME      1471982292697.31_20160823195812-00285.vert.dedup.parsed.tagged.mg4j
2 position      token  tag    lemma  parpos function      parword parlemma
    paroffset     link   length docuri lower  nerid  nertag param0 param1 param2
     param3 param4 param5 param6 param7 param8 param9 nertype nerlength
3 %%#DOC 5ab108d7-3992-50b7-be8c-d78077788443
4 %%#PAGE Justin Bieber Throws Up On Stage [VIDEO]  http://1061evansville.com/
    justin-bieber-throws-up-on-stage-video/
5 1      Justin NP     Justin 0      ROOT   0      0      0      0      0      0
         justin a:a7d4769f66  0      0      0      0      0      0      0      0
          0     0       0      0      0
6 2      Bieber NP     Bieber 1      SUFFIX Justin Justin -1     0      0      0
         bieber a:a7d4769f66  person http://en.wikipedia.org/wiki/Justin_Bieber
    http://athena3.fit.vutbr.cz/kb/images/freebase/0ncqgf3.jpg|http://athena3.fit.
    vutbr.cz/kb/images/wikimedia/commons/3/31/Believe_Tour_7,_2012.jpg
    Justin_Bieber M Stratford__Ontario__Canada 1994_03_01    0      0      0
    Canadian      kb     2
7 3      Throws NP     Throws 1      SUFFIX Justin Justin -2     0      0      0
         throws 0      0      0      0      0      0      0      0      0      0
          0     0       0      0
8 4      Up     NP     Up     1      SUFFIX Justin Justin -3     0      0      0
         up     0      0      0      0      0      0      0      0      0      0
          0     0       0      0
```

Manatee

Manatee je formát XML obsahující vertikální text určený pro indexaci nástrojem Manatee ze systému NoSketch Engine[15].

Příklad anotace v tomto formátu je následující:

```
1 <doc id="40d97753905b19a5-92a473a6bc257a5a166f94e5cacb8600" url="http://1061
    evansville.com/justin-bieber-throws-up-on-stage-video/" title="Justin Bieber
    Throws Up On Stage [VIDEO]">
2 <head>
3 <person nid="p:a54aa6295d" url="http://en.wikipedia.org/wiki/Justin_Bieber" image
    ="http://athena3.fit.vutbr.cz/kb/images/freebase/0ncqgf3.jpg|http://athena3.
```

[13] http://knot.fit.vutbr.cz/corpproc/corpproc_cs.html
[14] http://mg4j.di.unimi.it/
[15] https://nlp.fi.muni.cz/trac/noske

```
    fit.vutbr.cz/kb/images/wikimedia/d/d9/Believe_Tour_12,_2012.jpg" name="Justin
    Bieber" gender="M" birthplace="London" birthdate="1994-03-01" deathplace=""
    deathdate="" profession="" nationality="Canadian">
1   Justin  NP  Justin  0   ROOT    _   _   _   _   _   _
2   Bieber  NP  Bieber  1   SUFFIX  Justin  _   Justin  -1  _   _
</person>
3   Throws  NP  Throws  1   SUFFIX  Justin  _   Justin  -2  _   _
4   Up  NP  Up  1   SUFFIX  Justin  _   Justin  -3  _   _
5   On  IN  on  0   ROOT    _   _   _   _   _   _
6   Stage   NP  Stage   5   SUFFIX  On  _   on  -1  _   _
```

Manatee 2

Formát Manatee 2 je druhou verzí formátu Manatee určenou pro další experimenty skupiny KNOT. Od formátu Manatee se liší ve způsobu uložení anotací, přičemž je cílem snížit objem ukládaných dat.

Příklad anotace v tomto formátu je následující:

```
<doc id="40d97753905b19a5-92a473a6bc257a5a166f94e5cacb8600" url="http://1061
    evansville.com/justin-bieber-throws-up-on-stage-video/" title="Justin Bieber
    Throws Up On Stage [VIDEO]">
<head>
1   Justin  NP  Justin  0   ROOT    _   _   _   _   _   _
      p:a54aa6295d  _   _   _   _   _   _   _   _   _
      _   _   _   _
2   Bieber  NP  Bieber  1   SUFFIX  Justin  _   Justin  -1  _   _
      p:a54aa6295d  person http://en.wikipedia.org/wiki/Justin_Bieber http://
    athena3.fit.vutbr.cz/kb/images/freebase/0ncqgf3.jpg|http://athena3.fit.vutbr.
    cz/kb/images/wikimedia/d/d9/Believe_Tour_12,_2012.jpg Justin Bieber M London
    1994-03-01 _    _   _   Canadian    kb  2
3   Throws  NP  Throws  1   SUFFIX  Justin  _   Justin  -2  _   _
      _   _   _   _   _   _   _   _   _   _
      _   _   _
4   Up  NP  Up  1   SUFFIX  Justin  _   Justin  -3  _   _
      _   _   _   _   _   _   _   _   _   _
      _   _   _
5   On  IN  on  0   ROOT    _   _   _   _   _   _
      _   _   _   _   _   _   _   _   _   _
      _   _   _
6   Stage   NP  Stage   5   SUFFIX  On  _   on  -1  _   _
      _   _   _   _   _   _   _   _   _   _
      _   _   _
```

Kapitola 5

Návrh řešení

V této kapitole je popsán návrh programu pro efektivní práci se znalostní bází ve sdílené paměti (sekce 5.1) a komponenty pro sémantické obohacení textu (SEC) využívající sdílenou znalostní bázi a daný NER (sekce 5.2). Čtenář se zde dozví o možných překážkách při implementaci a jejich řešení.

5.1 Démon pro sdílení znalostní báze

Znalostní báze skupiny KNOT@FIT (popsaná v kapitole 3.1) je veliký objekt, který sám o sobě je problém držet v paměti. Cílem démona je, aby práce se znalostní bází byla co nejrychlejší a s co nejmenší spotřebou paměti. Z toho vyplývá, že data v paměti musí být strukturována tak, aby přístup na libovolný řádek byl nejlépe s velmi malou konstantní časovou složitostí. Dle požadavků je třeba počítat s tím, že démon bude využíván nástrojem NER, který je v jazyce Python (jak bylo zmíněno dříve v kapitole 2).

Doktor Dytrych mi navrhl dva způsoby, jak by démon mohl se znalostní bází pracovat:

1. Načíst celou znalostní bázi jako pole řetězců odpovídající řádkům ve znalostní bázi a na požadavek vrátit ukazatel do sdílené paměti na tento řetězec.

2. Jednotlivé řádky znalostní báze načíst jako struktury a na požadavek vrátit ukazatel do sdílené paměti a typ struktury.

Oba navržené způsoby počítají s tím, že démon bude vracet ukazatel do sdílené paměti. Je třeba počítat s tím, že ukazatel v rámci sdílené paměti může být pouze relativní k adrese, na které je namapován počátek sdílené paměti (jak jsem uvedl v kapitole 4.1). Z toho vyplývá, že každý program, který bude chtít démona využívat, si musí namapovat sdílenou paměť do svého virtuálního adresového prostoru a interpretovat získaný relativní ukazatel.

První způsob vyžaduje pouze strukturu s polem řetězců zakončených znakem „`\0`“ a délku tohoto pole. To předpokládá, že se v dokumentu nesmí vyskytovat znak „`\0`“.

Druhý způsob je značně složitější. Vyžaduje definici struktur, jenž se mohou v budoucnu změnit. Může se změnit počet atributů nebo typů. Abych vyřešil tuto komplikaci, specifikoval jsem hlavičku znalostní báze (KB), jenž slouží jako konfigurační soubor pro načítání samotné znalostní báze.

Další překážkou v druhém způsobu je složitost implementace dynamických struktur v jazyce C. Bude tedy jednoduší v jazyce C implementovat pouze rozdělení hlavičky KB a samotné znalostní báze na dvě tabulky s proměnlivými počty sloupců na řádcích. Na požadavek typu řádek a sloupec pak vrátit ukazatel na odpovídající řetězec ve znalostní bázi.

To znamená nechat typ a dynamickou strukturu na knihovnu implementovanou ve vyšším programovacím jazyce.

Při implementaci bude rovněž třeba počítat s tím, že minimální velikost alokovatelné sdílené paměti na operačních systémech GNU/Linux a architekturách x86_64 bývá 4 KiB (4096 bajtů)[1]. Proto by, pokud bychom alokovali každý řádek zvlášť, docházelo k nevyužití velké části alokované paměti, tedy ke ztrátám interní fragmentací. Ještě hůře by to dopadlo, kdybychom alokovali každý řádek i každý sloupec zvlášť. Z toho důvodu bude vhodné mít pouze jednu alokaci ve sdílené paměti na celou znalostní bázi.

5.1.1 Specifikace hlavičky KB

Aby mohla být znalostní báze (KB) používána nástroji, musí nástroje odněkud vzít informace o typech ve znalostní bázi. Pokud jsou tyto informace jako konstanty přímo v kódu nástrojů, pak při úpravě typů ve znalostní bázi musí být změněny i odpovídající konstanty. Proto jsem specifikoval hlavičku KB dle typů entit obsažených ve znalostní bázi a její syntax definoval v Backusově-Naurově formě (BNF):

```
<hlavička KB> ::= <řádek> "\n"
    | <řádek> <hlavička KB>

<řádek> ::= <první sloupec> "\n"
    | <první sloupec> "\t" <ostatní sloupce> "\n"

<první sloupec> ::= "<" <jméno typu> ">" <sloupec>
    | "<" <jméno typu> ":" <jméno podtypu> ">" <sloupec>

<ostatní sloupce> ::= <sloupec>
    | <sloupec> "\t" <ostatní sloupce>

<sloupec> ::= <název sloupce>
    | "{" <příznaky> "}" <název sloupce>
    | "{[" <prefix hodnoty> "]}" <název sloupce>
    | "{" <příznaky> "[" <prefix hodnoty> "]}" <název sloupce>
```

kde:

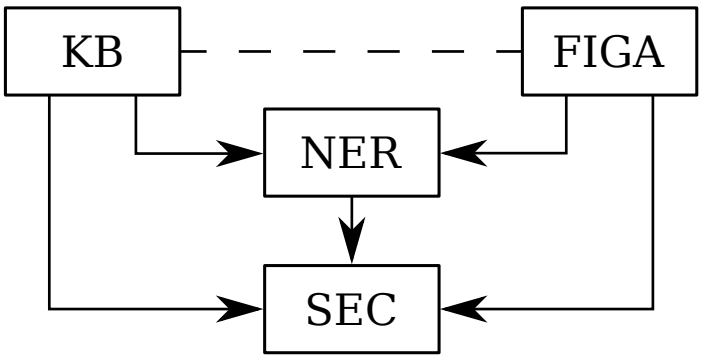
- `<jméno typu>` je řetězec označující typ,
- `<jméno podtypu>` je řetězec označující podtyp,
- `<příznaky>` jsou znaky, které určují datový typ (s – string, . . .), více hodnot v datech ve sloupci (m – multivalue) a identifikátor (i – identifier),
- `<prefix hodnoty>` je řetězec, který bude připojen k datům v tomto sloupci,
- `<název sloupce>` je řetězec se jménem sloupce.

```
příklad prvního sloupce v hlavičce: <location>ID
příklad prvního sloupce v datech: l:21931315183

příklad ostatních sloupců v hlavičce: {iu[http://en.wikipedia.org/]}WIKIPEDIA URL
příklad ostatních sloupců v datech: wiki/city_of_london
```

[1] Minimální velikost viz `/proc/sys/kernel/shmmni`

5.2 Komponenta pro sémantické obohacení textu

Komponenta pro sémantické obohacení textu (SEC) má využívat sdílenou znalostní bázi, navrženou v kapitole 5.1, nástroj Figa, popsaný v kapitole 3.2, a NER, jehož stav jsem popsal v kapitole 3.3. V obrázku 5.1 je znázorněno propojení mezi jednotlivými komponentami.



Obrázek 5.1: Propojení SEC a stávajících komponent. SEC využívá znalostní bázi, sdílenou pomocí démona pro znalostní bázi, a nástroje Figa a NER. Nástroj NER pro sebe načítá znalostní bázi a předpracovává ji. K vyhledání entit z textu ve znalostní bázi využívá nástroj Figa. Nástroj Figa k tomu používá slovníky vygenerované ze znalostní báze.

Vstupem SEC má být text pro zpracování nástrojem NER, popř. nástrojem Figa, zapouzdřený, společně s požadavky na výstup SEC (konfigurací), do formátu JSON (jak bylo uvedeno dříve v kapitole 2). Výstupem je pak dle nastavení pouze anotovaný text dokumentu, nebo celý dokument i s anotacemi v jednom z formátů definovaných v kapitole 4.7, popř. ve speciálním formátu SXML (Suggestion XML – viz dále, v sekci 5.2.3).

Navrhl jsem dvě možnosti, jak nástroj NER, popř. nástroj Figa, používat v SEC. První možností je spustit jej v subshellu[2] v démon módu, zadávat dokumenty na standardní vstup a v SEC zpracovávat standardní výstup nástroje NER. Nevýhodou tohoto řešení je, že démon mód poskytuje pouze sekvenční zpracování, ale SEC má umožnit vícevláknové zpracování. Možností druhou je využít pro SEC jazyk, ve kterém je napsán NER, a importovat jej jako modul. Odpadá tak nutnost zpracovávat standardní výstup nástroje NER a je možné pracovat přímo s daty. Jelikož nástroj NER může obsahovat chyby, které by mohly způsobit pád SEC, bude vhodné odchytávat u něj signály a popř. jej volat jako podproces. Jiné nástroje NER však mohou být v jiném programovacím jazyce, proto musí být zpracovávání standardního výstupu nástroje NER implementováno.

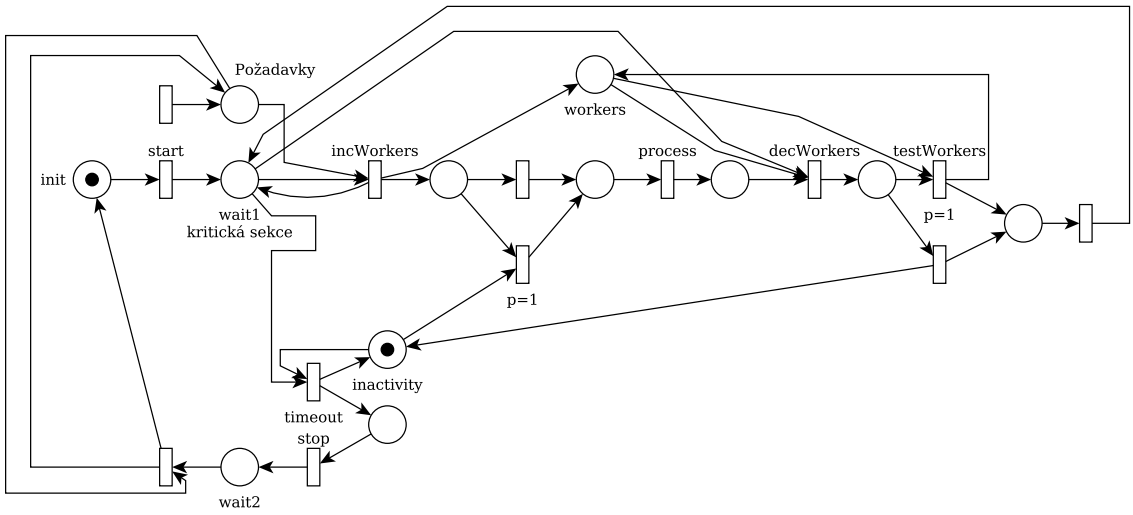
Nástroj NER neposkytuje vícevláknové zpracování a jeho běh vyžaduje spoustu paměti, protože před každým během načítá znalostní bázi a předzpracovává si ji. SEC tedy nemůže NER spouštět jako instanci, protože by došlo k jednomu z následujících případů:

- Vlákna SEC by sdílela jednu instanci nástroje NER, čímž by nebylo využito potenciálu vícevláknového zpracování (démon mód nástroje NER poskytuje pouze sekvenční zpracování).
- Více instancí nástroje NER by rychle zahltilo operační paměť, protože NER nevyužívá sdílené paměti a každá instance má navíc předzpracovaná data ze znalostní báze.

[2]Potomek spuštěný pomocí shellu.

Je tedy nutné přepsat NER alespoň do takové míry, aby využíval sdílenou znalostní bázi. Aby se sdílela předzpracovaná data ze znalostní báze, bude jednodušší využít druhé možnosti integrace nástroje NER, popsané v předchozím odstavci, tedy SEC napsat v jazyce Python 2.7.x a NER importovat jako modul.

Vyjdu-li z nástroje NER skupiny KNOT@FIT, mohu předpokládat, že jiné nástroje NER budou také zabírat mnoho paměti. Bylo by tedy vhodné, aby SEC pamětí šetřil a nedržel v ní zbytečně data aktuálně nepoužívaných nástrojů NER. Proto jsem navrhl pomocí Petriho sítě objekt starající se o vlákna zpracovávající požadavky na nástroje NER (viz obrázek 5.2).

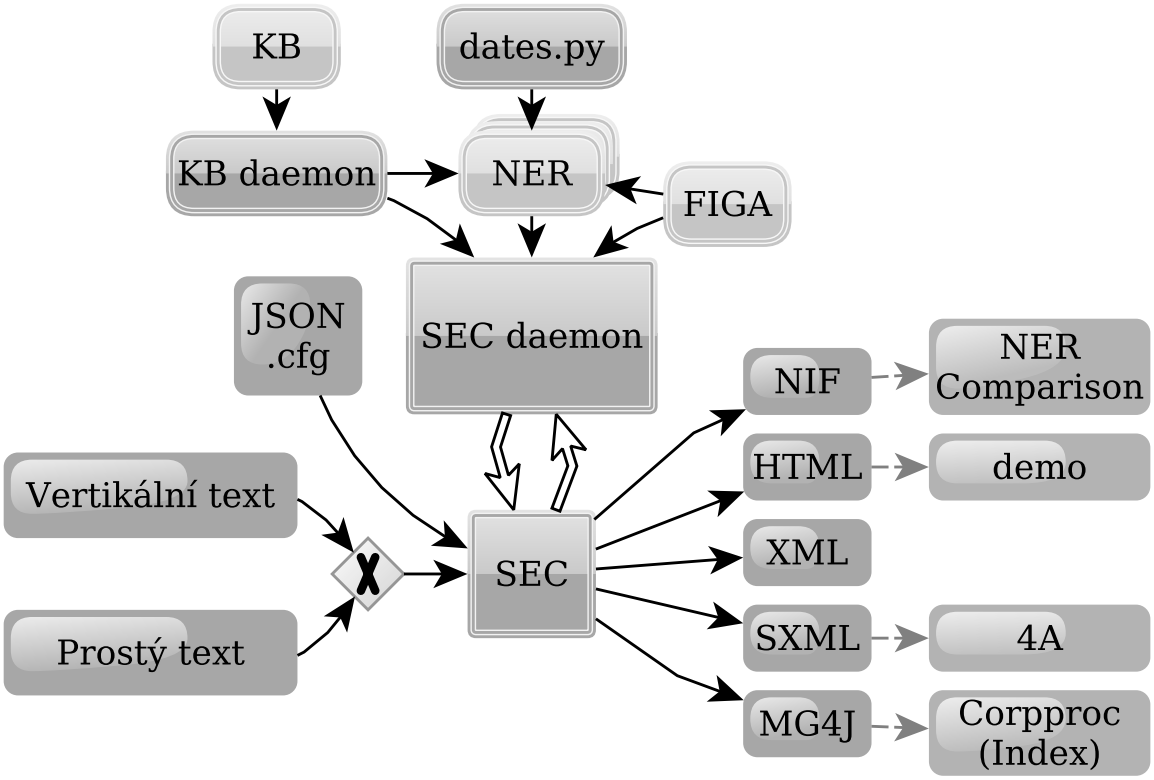


Obrázek 5.2: Petriho síť znázorňující zpracování požadavků v SEC pro libovolný nástroj NER s uvolněním zdrojů při nečinnosti. Nástroje NER jsou při spuštění (`init`) načteny do paměti (přes `start`). Následuje čekání na požadavky (`wait1`). Každému příchozímu požadavku je přiřazeno (přes `incWorkers`) vlákno (`worker`), ve kterém je požadavek zpracován a po zpracování je vlákno uvolněno (přes `decWorkers`). Pokud žádný požadavek není zpracováván (`inactivity`) a na požadavky se čeká již dlouho, pak (přes `timeout`) dojde k uvolnění nečinného nástroje NER z operační paměti (přes `stop`). K opětovnému načtení do paměti dojde až s příchodem dalšího požadavku (na něj se čeká ve `wait2`).

Umožnit využívat SEC z příkazové řádky a zároveň přes protokol HTTP nebude nikterak velikým problémem, protože SEC bude pracovat ve více vláknech. Pro využití SEC přes protokol HTTP stačí využít metody POST podobně jako příkazového řádku. Tedy přes metodu POST poslat data ve formátu JSON obsahující název služby, její atributy a případně i text určený k anotaci.

Pokud má SEC umožnit vícevláknové zpracování dotazů, je otázkou, jakým způsobem se k němu může dostat více dotazů. Přes protokol HTTP to lze pomocí konkurenčního HTTP serveru. Aby bylo možné poslat přes standardní vstup příkazového řádku více dotazů paralelně, je třeba nástroj SEC rozdělit na serverovou a klientskou část. Klienti budou dotazy posílat serveru – *SEC démonu*. Server je bude zpracovávat a posílat odpovědi zpět klientům. Stačí, aby komunikace probíhala skrze *unixové sokety* (UDS) a jednoduchý komunikační protokol, jenž je popsán dále v sekci 5.2.2. HTTP server by mohl být implementován též jako klient přistupující k *SEC démonu*. Kompletní návrh je také možné spatřit na obrázku 5.3.

Nyní rozeberu rozšíření označené jako *testovací mód*, jenž umožní testovat práci se strukturovanými anotacemi, které NER neumí vytváře (jak je popsáno v kapitole 2). Testovací mód vyžaduje volbu adresáře, ve kterém SEC provede vyhledání souborů, jenž výsledek získaný nástrojem NER obohatí či nahradí. Název souboru očekává v atributu dotazu na službu anotace textu, jenž je zadán uživatelem. Uživatel si tak může zvolit soubor, který mu bude přečten. Zde je potenciální bezpečnostní riziko při špatné implementaci. Přidáme-li k tomu požadavek na volbu testovacího adresáře klientem, riziko se značně zvýší a je již na straně návrhu. Volbu testovacího adresáře klientem můžeme umožnit jen za předpokladu, že nám klient bude na vyzvání otevírat soubory, o které žádá, a posílat otevřené *popisovače souborů* skrze již zmíněné unixové sokety „SEC démonu“. Tím se ověří, že má přístup k souborům, o které žádá.



Obrázek 5.3: Kompletní návrh komponenty pro sémantické obohacení textu (SEC). Nástroj SEC je rozdělen na dvě části: „SEC daemon“ – server a „SEC“ – klient. „SEC daemon“ je jádrem celého SEC. Umožňuje paralelní vyřizování dotazů na jednom stroji v jedné instanci, čímž šetří paměť. „SEC“ jako klient zhotoví dotaz dle vstupu a konfigurace a odešle na „SEC daemon“. Ten dotazu přidělí vlákno, zpracuje jej a výsledek vrátí klientovi. Vstup může být zadán z příkazové řádky, ale zároveň i přes protokol HTTP, a to konkurenčně. Výstup je samozřejmě odeslán tam, odkud přišel vstup. Na tomto obrázku je na výstupu „SEC“ znázorněn výběr výstupních formátů a případně jejich využití.

5.2.1 Specifikace poskytovaných služeb

Komponenta pro sémantické obohacení textu bude poskytovat řadu služeb. Základem bude služba anotace textu, dále služba vyhledání entity dle jejího názvu a služba pro výpis všech

typů entit, jenž mohou být ve znalostní bázi obsaženy, včetně jejich atributů. Požadavky na služby budou očekávány ve formátu JSON. Tyto služby jsem již částečně popsal v cílech práce (kapitola 2). Zde popíši zmíněné služby podrobněji pro potřeby implementace a doplním i další služby, jenž se principiálně budou využívat i uvnitř komponenty pro sémantické obohacení textu.

Služby anotace textu

Cílem služeb anotace textu je poskytnout anotace k dokumentu zadanému na vstupu. Původní návrh služby anotace textu, jenž je obsažen v technické zprávě pro projekt Decipher [1], počítá se vstupními dokumenty v prostém textu a s výstupními formáty Text, Index, XML, HTML a RDF. V uvedené technické zprávě je tato služba pojmenována `annotate`. Návrh umožňuje skrze atributy ke službě `annotate` vybrat jeden a více výstupních formátů na jeden požadavek atributem `annotation_format`, povolit či zakázat výběr nejpravděpodobnějšího významu anotačním nástrojem u anotovaného textu atributem `disambiguate` a omezit anotace na typy entit a jejich atributy dle atributu služby `types_and_attributes`. Vstupní dokument v prostém textu očekává v atributu `input_text`.

Pro kompatibilitu s původním návrhem jsem výstup více než do jednoho výstupního formátu vyřešil doplněním znaku „`,`“ a novým řádkem „`\n`“ mezi jednotlivé výstupní formáty, což pro upřesnění popíši i v Backusově-Naurově formě (BNF):

```
<více výstupních formátů> ::= <výstupní formát v JSON>
    | <výstupní formát v JSON> "," <nový řádek> <více výstupních formátů>
```

Omezení anotace pomocí atributu `types_and_attributes` půjde provést několika způsoby. Bude možné povolit jednotlivé typy entit a k nim buď všechny jejich atributy (syntaxe „`{ str(type): "all"}`“) nebo jen některé (syntaxe „`{ str(type): [ str(attribute), ... ] }`“). Výchozí hodnotou bude „`"all"`“, což znamená, že je povolen výpis všech typů entit i všech jejich atributů.

Protože komponent pro sémantické obohacení textu musí umožnit i použití jiných nástrojů NER, upravil jsem původní návrh služby `annotate` a doplnil jsem do něj atribut `enrichment_engine` pro výběr nástroje NER a atribut `enrichment_engine_timeout` pro omezení délky zpracování zvoleným nástrojem NER. Dále jsem jej doplnil o výstupní formáty NIF, Index 2 a SXML. Pro výstupní formát NIF bude třeba zadat adresu (URL), ze které byl dokument přebrán (atribut `document_uri`). Aby bylo možné zpracovávat dokumenty ve vertikálním textu, specifikoval jsem pro ně službu s názvem `annotate_vertical`. Zejména proto, aby nebylo třeba rozpoznávat formát vstupního dokumentu, a také vzhledem k odlišnostem mezi možnými výstupními formáty pro prostý text a vertikální text. Zapouzdření výstupu služby `annotate` do formátu JSON nemusí být vždy žádoucí. Může zdržovat při dalším zpracování a překážet v postupném výstupu. Z těchto důvodů jsem doplnil atribut `plaintext`. V případě využití tohoto atributu bude více výstupních formátů zadaných atributem `annotation_format` odděleno znakem „`\0`“.

Na následující řádcích popíši syntax požadavků a odpovědí pro služby anotace textu. Pro popis syntaxe jsem zvolil podobný formát jako je v původním návrhu. Rozdíl je v tom, že jsem typ vyjádřil přímo. Jako první uvedu syntax služby `annotate`:
Formát požadavku služby ***`annotate`****:*

```
{
    "annotate": {
        "input_text": str,
```

```
        "annotation_format": [ str, ... ],
        "disambiguate": int,
        "document_uri": str,
        "types_and_attributes": "all" | { str(type): "all" } | { str(type): [ str(
            attribute), ... ] },
        "enrichment_engine": str,
        "enrichment_engine_timeout": int,
        "plaintext": bool
    }
}
```

kde:

- Volitelnými atributy budou: `disambiguate`, `types_and_attributes`, `document_uri`, `enrichment_engine`, `enrichment_engine_timeout` a `plaintext`.
- Atribut `document_uri` bude povinný, pokud mezi hodnotami atributu `annotation_format` bude výstupní formát NIF.

Formát odpovědi služby `annotate`:

```
{
    "annotation": str
}
```

Komponent pro sémantické obohacení bude pro službu `annotate` používat nástroj NER. Nebude tedy problém vytvořit službu s názvem `get_raw_annotations`, jenž bude poskytovat výstup z nástroje NER. Služba `get_raw_annotations` bude mít následující syntax:
Formát požadavku služby `get_raw_annotations`:

```
{
    "get_raw_annotations": {
        "input_text": str,
        "disambiguate": int,
        "enrichment_engine": str,
        "enrichment_engine_timeout": int,
        "plaintext": bool
    }
}
```

kde:

- Volitelnými atributy budou: `disambiguate`, `enrichment_engine`, `enrichment_engine_timeout` a `plaintext`.

Formát odpovědi služby `get_raw_annotations`:

```
{
    "annotation": str
}
```

Již dříve zmíněná služba `annotate_vertical` bude sloužit k anotaci vertikálních dokumentů. K tomu bude potřebovat implementovat převod z vertikálního textu na prostý text, aby jej bylo možné anotovat nástrojem NER. Tento převod bude poskytnut službou `deverticalize`. Služba `annotate_vertical` bude mít následující syntax:
Formát požadavku služby `annotate_vertical`:

```
{
    "annotate_vertical": {
        "input_text": str,
        "annotation_format": str,
        "vert_in_cols": [ str, ... ],
        "vert_out_cols": [ str, ... ],
        "types_and_attributes": "all" | { str(type): "all" } | { str(type): [ str(
            attribute), ... ] },
        "enrichment_engine": str,
        "enrichment_engine_timeout": int,
        "filename": str,
        "num_workers": int,
        "plaintext": bool,
        "max_values_per_col": int | null,
        "wiki_mode": bool,
        "enable_figa": bool
    }
}
```

kde:

- Volitelnými atributy budou: `vert_in_cols`, `vert_out_cols`, `types_and_attributes`, `enrichment_engine`, `filename`, `enrichment_engine_timeout`, `num_workers` a `plaintext`.
- Atribut `annotation_format` bude určovat výstupní formát. Možnými formáty budou MG4J, Manatee, Manatee 2 a ElasticSearch.

Formát odpovědi služby `annotate_vertical`:

```
{
    "annotation": str | [
        {
            "title": str,
            "uri": str,
            "article": str
        },
        ...
    ]
}
```

Služba `deverticalize` má atribut `vert_in_cols`, kterým se v případě vícesloupcového formátu vybírají sloupce obsahující samotný vertikální text. Tato služba bude mít následující syntax:
Formát požadavku služby `deverticalize`:

```
{
    "deverticalize": {
        "input_text": str,
        "vert_in_cols": [ str, ... ]
    }
}
```

Formát odpovědi služby `deverticalize`:

```
{
    "deverticalized": [
```

```
        {
            "id": str,
            "document": str
        },
        ...
    ]
}
```

Služby pro hledání entit ve znalostní bázi

Pro hledání entit ve znalostní bázi je v původním návrhu definována služba `get_entities`, jejímž úkolem je vyhledat entity dle názvu nebo prefixu názvu (doplněného znakem „*“) zadaného atributem `input_string`. Oproti původnímu návrhu jsem odstranil atribut `type`, jenž je volitelný a měl omezit hledání na určitý typ entity. Tento účel je velmi podobný atributu `types_and_attributes` (jenž je v původním návrhu vyžadován) a proto ním byl nahrazen atribut `type`. Dalším volitelným atributem `max_results` se určuje maximální počet entit ve výstupu.

Doplnil jsem také službu podobného charakteru s názvem `get_entity_by_uri`. Ta vyhledá entitu dle odkazu (URI), jenž by měl být v rámci znalostní báze jedinečný.

Formát požadavku služby `get_entities`:

```
{
    "get_entities": {
        "input_string": str,
        "types_and_attributes": "all" | { str(type): "all" } | { str(type): [ str(
            attribute), ... ] },
        "max_results": int
    }
}
```

Formát odpovědi služby `get_entities`:

```
{
    "data": [
        {
            str(type): {
                str(attribute): str,
                ...
            }
        },
        ...
    ]
}
```

Formát požadavku služby `get_entity_by_uri`:

```
{
    "get_entity_by_uri": {
        "input_string": str,
        "types_and_attributes": "all" | { str(type): "all" } | { str(type): [ str(
            attribute), ... ] }
    }
}
```

Formát odpovědi služby `get_entity_by_uri`:

```
{
    "data": [
        {
            str(type): {
                str(attribute): str,
                ...
            }
        },
        ...
    ]
}
```

Ostatní služby

V původním návrhu je definována služba pro výpis všech dostupných typů entit a jejich atributů, která má název `get_entity_types_and_attributes`. Aby bylo zřejmé, ze kterých nástrojů NER je možné si vybrat a zvolit jako hodnotu pro atribut `enrichment_engine` u služeb anotace textu, navrhl jsem navíc službu s názvem `get_enrichment_engines`, jenž vypíše dostupné nástroje NER. Dále jsem přidal službu pro zjištění aktuální verze znalostní báze `get_kb_version`.

Formát požadavku služby `get_entity_types_and_attributes`:

```
{
    "get_entity_types_and_attributes": {}
}
```

Formát odpovědi služby `get_entity_types_and_attributes`:

```
{
    "data": [
        {
            "type": str(type),
            "attributes": [
                str(attribute),
                ...
            ]
        },
        ...
    ]
}
```

Formát požadavku služby `get_enrichment_engines`:

```
{
    "get_enrichment_engines": {}
}
```

Formát odpovědi služby `get_enrichment_engines`:

```
{
    "enrichment_engines": [ str, ... ]
}
```

Formát požadavku služby `get_kb_version`:

```
{
    "get_kb_version": {}
}
```

Formát odpovědi služby `get_kb_version`*:*

```
{
    "version": int
}
```

5.2.2 Specifikace vnitřního komunikačního protokolu

Pro komunikaci mezi *SEC démonem* – serverem a *SEC klientem* je třeba navrhnout jednoduchý protokol. Musí umožnit přenášet velké množství dat a bude vhodné, aby umožnil přenášet otevřený *popisovač souboru*. K tomu jsou vhodné *unixové sokety* (UDS) ve spojovaném módu pro přenos toku bajtů (`SOCK_STREAM`). Při něm unixové sokety garantují spolehlivé doručování ve správném pořadí podobně, jak je tomu u protokolu TCP. Navíc unixové sokety umožňují přenášet otevřený popisovač souborů mezi procesy. Návrh průběhu komunikace je tedy následující:

1. Server čeká na klienty.
2. Klient se připojí.
3. Klient pošle nastavení serveru (adresář k testovacímu módu a JSON s nastavením požadované služby).
4. Server přijme a potvrdí klientovi.
5. Klient pošle data ke zpracování serveru (pokud je požadovaná služba nevyžaduje, mohou být o nulové délce).
6. Server po klientovi v rámci testovacího módu může žádat o otevření několika souborů a poslání jejich popisovače souboru (tím se také prokáže oprávnění klienta soubor otevřít). V takovém případě si klient ověří, že se jedná o soubor, jehož název předal v konfiguraci, otevře jej a odešle otevřený popisovač souboru serveru.
7. Server pošle klientovi zpracovaná data.
8. Klient končí spojení nebo pokračuje bodem č. 5, popř. bodem č. 3.

Pokud server zjistí chybné nastavení nebo dojde k chybě při zpracování, pošle klientovi informace o chybě a ukončí s ním spojení.

Příkazy a struktury paketů

Příkazy:

- `set_config` – odesílání konfigurace v JSON
- `set_testing_dir` – odesílání cesty k testovacímu adresáři (mělo smysl, dokud nebyl odesílán otevřený popisovač souboru)
- `process_data` – odesílání vstupního dokumentu ke zpracování
- `wrought_data` – odesílání výsledku
- `open_file_thru_client` – požadavek na otevření souboru klientem

- `acknowledgment` – potvrzení
- `error` – chyba

Každý paket má dynamicky generované dvoumístné číslo Opcode. Pro příkazy se používají dvě struktury paketů. Pro chyby a potvrzení je to:

2 bajty	Řetězec	2 bajty
`Opcode`	`Chybová zpráva`	`CRLF`

Pro zbytek (vyjma odesílání otevřených popisovačů souborů) je to struktura, která se opakuje, dokud není počet bajtů dat roven nule:

2 bajty	Číslo (desítkové)	2 bajty	N bajtů	2 bajty
`Opcode`	`Počet bajtů dat N`	`CRLF`	`Raw data`	`CRLF`

Pro odesílání otevřených popisovačů souborů se používá knihovna `fdsend`.

5.2.3 Specifikace výstupního formátu SXML

Jak již bylo uvedeno v cílech (kapitola 2), bylo třeba navrhnout formát Suggestion XML, neboli SXML, jenž je aplikací univerzálního značkovacího metajazyka XML a obsahuje pouze anotovaný text, protože je určen pro nástroje, které pracují s textem a jeho anotacemi odděleně. Strukturu formátu SXML jsem navrhl v jazyce pro popis struktury dokumentů v jazyce XML – DTD:

```
<?xml encoding="UTF-8"?>
<!ELEMENT suggestion (text*)>

<!ELEMENT text (annotation*)>
<!ATTLIST text
  s_offset CDATA #REQUIRED
  e_offset CDATA #REQUIRED
  string CDATA #REQUIRED>

<!ELEMENT annotation (attribute*)>
<!ATTLIST annotation
  id CDATA #IMPLIED
  type NMTOKEN #REQUIRED>

<!ELEMENT attribute (#PCDATA)>
<!ATTLIST attribute
  annotType NMTOKEN #IMPLIED
  name NMTOKEN #REQUIRED
  type (string|decimal|date|image|integer|uri) #REQUIRED>
```

Formát SXML bude využíván především anotačním nástrojem 4A[3]. Tento nástroj umožňuje vytváření jednoduchých i strukturovaných anotací, přičemž uživatel může anotace vytvářet nejen manuálně, ale i schvalováním nabídek generovaných pomocí nástroje NER v pozadí. Když si anotační server načetl znalostní bázi do paměti ve vlastní režii, byla spotřeba paměti větší než 200 % velikosti KB, což bylo s rostoucí velikostí KB neudržitelné, a NER byl

[3] http://knot.fit.vutbr.cz/annotations/

spouštěn v démon módu, což vylučovalo paralelní zpracování. Proto bude anotační server využívat SEC umožňující paralelní zpracování s minimem režií. Formát SXML byl navržen tak, aby bylo možno využívat již existujících funkcí pro zpracování XML v anotačním serveru (anotační nástroj interně využívá 2 formáty anotací, které jsou aplikací XML – 4A[4] a Open Annotation[5]) a současně aby byly minimalizovány režie při přenosu velkého množství alternativních anotací. Proto je každý úsek anotovaného textu uveden pouze jednou a pro něj jsou uvedeny jednotlivé alternativní anotace (na rozdíl od formátů 4A a Open Annotation, které mají anotovaný text v každé anotaci samostatně).

Příklad anotace ve formátu SXML:

```
<suggestion xmlns:geo="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <text s_offset="0" e_offset="24" string="James 'Big Jim' Larkin">
    <annotation type="Person"
                id="http://localhost:8080/Annotations/annotations/337">
      <attribute name="Visual_representation" type="Image">
        http://athena3.fit.vutbr.cz/kb/images/freebase/02b_lmj.jpg
      </attribute>
      <attribute name="Name" type="String">James Larkin</attribute>
      <attribute name="Disambiguation" type="String">
        Irish trade union leader and socialist activist
      </attribute>
      <attribute name="Description" type="Text">
        <![CDATA[James Larkin was an Irish trade union leader ...]]>
      </attribute>
      <attribute name="URI_of_entity" type="URI">
        http://www.freebase.com/m/03msv5
      </attribute>
      <attribute name="Wikipedia_URI" type="URI">
        http://en.wikipedia.org/wiki/James_Larkin
      </attribute>
    </annotation>
  </text>
</suggestion>
```

5.2.4 Specifikace výstupu nástrojů NER

Aby bylo možné integrovat i jiné nástroje NER, specifikoval jsem v Backusově-Naurově formě (BNF) jejich výstup dle implementace stávajícího nástroje NER [9] (výstup nástroje NER skupiny KNOT@FIT měl dosud specifikaci pouze slovní). U výstupu z nástrojů NER předpokládám následující syntaxi:

```
<výstup z NER> ::= <origin_base>
    | <origin_base> "\t" <id>
    | <origin_base> "\t" <id> "\t" <direct_attributes>
<origin_base> ::= <start_offset> "\t" <end_offset> "\t" <data_type>
      "\t" <string_between_offsets> "\t" <data>
<data_type> ::= "kb"
    | "activity"
```

[4] http://knot.fit.vutbr.cz/annotations/4A_protocol_1_1_en.html
[5] http://www.openannotation.org/spec/core/

```
    | "date"
    | "interval"
    | "coref"
    | "uri"
<data> ::= <data-kb>
    | <data-activity>
    | <data-date>
    | <data-interval>
    | <data-coref>
    | <data-uri>
<data-kb> ::= <KB_row> | <KB_row> ";" <data-kb>
<data-date> ::= <year> "-" <month> "-" <day>
<data-interval> ::= <data-date> " -- " <data-date>
<data-coref> ::= <data-kb>

<direct_attributes> ::= <attribute> | <attribute> "|" <direct_attributes>
<attribute> ::= <attribute_name> "[" <attribute_type> "]=" <attribute_value>
<attribute_type> ::= "string" | "decimal" | "date"
    | "image" | "integer" | "uri" | <other_attribute_type>

<year> ::= <digit> <digit> <digit> <digit>
<month> ::= <digit> <digit>
<day> ::= <digit> <digit>
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

kde:

- `<start_offset>` a `<end_offset>` vyjadřují počáteční a koncovou pozici entity ve vstupním textu,
- `<data_type>` označuje typ entity,
- `<string_between_offsets>` je textová podoba entity ze vstupního textu,
- `<data>` jsou data podle `<data_type>`, tedy `<data-<data_type>>` (viz níže),
- `<id>` je identifikátor anotace
  - unikátní v rámci výstupu z NERu,
  - může být cokoliv (např. URI, číslo, prázdný řetězec),
  - má efekt pouze ve výstupu do SXML,
  - je-li prázdný řetězec, pak je ignorován.
- `<direct_attributes>` jsou atributy extra přidané nástrojem NER oddělené „`|`“,
  - většinou URI, číslo nebo řetězec bez speciálních znaků,
  - nesmí se v nich vyskytovat znaky „`|`“, „`\t`“ a „`\n`“,
  - využijí se pouze ve výstupu do SXML.
- `<attribute_type>` – vychází z XSD[6], může ale obsahovat i jiné typy, které jsou popsány v dokumentaci nástroje 4A[7] (např. typ „AnnotationLink“)
  - „string“ – řetězec,
  - „decimal“ – číselná hodnota,

[6] Viz https://cs.wikipedia.org/wiki/XML_Schema_Definition a https://www.w3.org/TR/xmlschema-2/#built-in-datatypes

[7] http://knot.fit.vutbr.cz/annotations/4A_protocol_1_1_en.html

– „date“ – datum ve formátu YYYY-MM-DD,
– „image“ – URI obrázku,
– „integer“ – celočíselná hodnota,
– „uri“ – URI obecného zdroje,
– ...

V případě, že `<data_type>` je:

- `"kb"`:
  pak `<data>` jsou čísla řádků ze znalostní báze oddělená „`;`“,
- `"activity"` – značí, že `<string_between_offsets>` je sloveso:
  pak `<data>` je infinitiv slovesa `<string_between_offsets>`,
- `"date"`:
  pak `<data>` je datum ve formátu ISO 8601 `YYYY-MM-DD`,
- `"interval"`:
  pak `<data>` jsou dvě data ve formátu ISO 8601 `YYYY-MM-DD` oddělená `" -- "`,
- `"coref"` – značí, že `<string_between_offsets>` je zájmeno:
  pak `<data>` jsou čísla řádků ze znalostní báze oddělená „`;`“,
- `"uri"`:
  pak `<data>` je URI na Wikipedia, Freebase, DBpedia nebo podobné.

5.3 Vylepšení nástroje NER

Pomalou inicializaci nástroje NER (popsaného v kapitole 3.3) částečně řeší propojení sdílené znalostní báze uvedené v předchozí sekci 5.2 této kapitoly. Dalším zdržením je předzpracování dat ze znalostní báze. Předzpracovaná data jsou konstantní pro každý běh nástroje NER a liší se pouze pokud se změní obsah znalostní báze. Nejjednodušším řešením tedy bude serializace předzpracovaných dat při prvním běhu se znalostní bází. Nedojde-li v dalším běhu ke změně znalostní báze, jsou předzpracovaná data načtena (deserializována) ze souboru. V opačném případě budou data předzpracována znovu.

Rozlišovací schopnost nástroje NER závisí mimo jiné na tom, jak zvládá využívat kontext. Vytvoření a integrace nástroje pro hledání dat[8] ve vstupním textu je proto důležité. Vzhledem k nástroji NER bude pro implementaci vhodné použít taktéž jazyk Python 2.7.x, aby mohl být importován jako modul. Dále to bude vhodné i vzhledem k nástrojům, jenž jazyk Python nabízí pro zpracování řetězců. Přesněji jde o rozšířené regulární výrazy podobné PCRE[9], které mohou být přehledné i při tvorbě komplexního regulárního výrazu. Pro převod dat do formátu ISO 8601 (formát rok-měsíc-den, např. „1999-12-28“) může posloužit modul *dateutil*, který zvládá různé anglické formáty data (např. „Sept. 12, 2007“ nebo „Jul 18 ’10“). Stačí se tedy zaměřit na tvorbu kompletního vyhledávacího regulárního výrazu. Kromě prostého data se v textu často vyskytují i intervaly (např. „June 6-Sept. 23, 2007“) a nepřesná data (např. „mid-19th century“ či „mid-1987“). Ty je třeba vhodně upravit na dvě data vymezující interval nebo nepřesné datum.

[8] „Hledáním dat“ se zde myslí hledání časového určení události složeného z čísla roku, případně i z označení měsíce a dne.
[9] Perl Compatible Regular Expressions https://www.pcre.org/

Kapitola 6

Implementace komponenty pro sémantické obohacení a jejích podčástí

Tato kapitola popisuje implementaci návrhů uvedených v předchozí kapitole. Nejprve bylo třeba, abych implementoval program pro efektivní práci se znalostní bází ve sdílené paměti. K tomu jsem napsal knihovnu a pro ni jsem vytvořil modul pro jazyk Python. Souběžně s implementací komponenty pro sémantické obohacení textu (SEC) jsem vylepšoval modul a nástroj pro rozpoznání pojmenovaných entit v textu (NER), který jsem s využitím již zmíněného modulu vylepšil tak, aby využíval sdílenou znalostní bázi. Implementoval jsem modul pro nástroj NER, jenž každé datum, které nalezne, poskytne nástroji NER ve struktuře s místem nalezení v textu a reprezentací toho data ve formátu ISO 8601. Po dokončení nástroje SEC jsem přepsal některé nástroje, jenž upravují znalostní bázi (KB), aby využívaly údajů uvedených v hlavičce znalostní báze.

6.1 Implementace nástrojů pro sdílení znalostní báze

Dle cílů (kapitola 2) a návrhu (kapitola 5.1) jsem implementoval démon pro sdílení znalostní báze, knihovnu a demonstrační program. Vytvořil jsem hlavičku k aktuální znalostní bázi dle specifikace Ing. Otrusiny [4] a uložil ji do souboru s názvem `HEAD-KB`.

Démon pro sdílení znalostní báze jsem implementoval v jazyce C. K tomu jsem využil systémové knihovny a principy popsané v kapitole 4.1. Obě verze zmíněné v návrhu jsem nejprve implementoval do té míry, aby je bylo možné porovnat, a následně jsem už vyvíjel pouze lepší verzi. Lépe vyšla verze 2, tedy verze, kde je znalostní báze načtena jako tabulka s proměnlivými počty sloupců na řádcích (průběh a výsledek testů bude podrobněji popsán dále, a to v kapitole 7.1).

Jelikož hlavička i znalostní báze se může časem měnit, abych předešel omylům, vyžaduji pro démona spojení hlavičky se znalostní bází. A aby bylo možné zjistit, která verze je aktuálně sdílena, vyžaduji i připojení čísla verze. Znalostní báze určená pro démona tedy musí odpovídat následující syntaxi v Backusově-Naurově formě (BNF):

```
<KB pro démon> :== <verze KB> <hlavička KB> <KB>
    | <hlavička KB> <KB>

<verze KB> :== "VERSION=" <číslo verze> "\n"
```

kde:

- `<číslo verze>` je počet sekund od epochy získaný příkazem `date +%s`,
- `<hlavička KB>` odpovídá BNF z kapitoly 5.1.1 (mezi řádky hlavičky a řádky znalostní báze tak vznikne volný řádek),
- `<KB>` je znalostní báze, jejíž význam sloupců odpovídá hlavičce (každý její řádek musí končit znakem „`\n`“).

Atributy démona jsou znalostní báze spojená s hlavičkou a verzí nebo její předzpracovaná podoba (kopie paměti dříve načtené znalostní báze) a případně i specifikace jména objektu ve sdílené paměti. Předzpracovaná podoba znalostní báze (vytvořená kopií paměti) slouží k urychlení příštího startu démona a neměla by se přenášet mezi různými architekturami. Průběh činnosti démona je takovýto:

1. Spuštění démona.
2. Kontrola předzpracované podoby znalostní báze. Je-li novější než znalostní báze zadaná atributem (nebo je-li předzpracovaná podoba znalostní báze zadaná atributem), zkopíruje se do sdílené paměti a skočí se na bod 6.
3. Postupné načtení a předzpracování znalostní báze (rozdělení na řádky a sloupce).
4. Načtení předzpracované znalostní báze do sdílené paměti.
5. Binární kopie sdílené paměti do souboru (k cestě ke znalostní bázi zadané atributem se přidá přípona „`.bin`“) pro urychlení dalšího spuštění (bod 2).
6. Výpis informace, že znalostní báze je připravena ve sdílené paměti.
7. Čekání na jeden ze tří ukončovacích signálů (`SIGINT`, `SIGTERM` a `SIGQUIT`).
8. Uvolnění jména objektu ve sdílené paměti a ukončení démona.

Knihovnu pro využití znalostní báze sdílené démonem jsem implementoval taktéž v jazyce C. To umožnilo převzít podstatnou část z démona a implementovat pouze funkce pro připojení a odpojení sdílené znalostní báze, funkci pro přístup k řádku a sloupci specifikovanému parametrem a funkci pro zjištění verze znalostní báze ve sdílené paměti.

Demonstrační program jsem implementoval v jazyce Python 2.7.x. Jeho účelem bylo porovnat dvě navržené verze démona a zároveň demonstrovat propojení jazyku Python s knihovnou napsanou v jazyce C. Posléze jsem implementoval i modul pro jazyk Python 2.7.x, který zabaluje knihovnu do třídy `KB_shm` pro její snadnější použití. Pro kompilaci knihovny i démona jsem vytvořil jediný `Makefile`.

6.2 Implementace komponenty pro sémantické obohacení

Komponentu pro sémantické obohacení textu jsem implementoval dle návrhu (kapitola 5.2) v jazyce Python 2.7.x. Rozdělil jsem ji do tří spustitelných skriptů. Skript `sec_daemon.py` odpovídá SEC démonu v návrhu a je tak jádrem komponenty pro sémantické obohacení. Skript `sec.py` je klientem k SEC démonu. Za svůj běh umožňuje zpracovat pouze jeden požadavek, tedy přijmout ze standardního vstupu, odeslat (s konfigurací) SEC démonu, přijmout odpověď a vytisknout na standardní výstup. To umožňuje zadat zvlášť konfiguraci (přes argument příkazového řádku) a text ke zpracování (přes standardní vstup). Třetím

skriptem je `sec_api.py`, jenž vytváří konkurenční HTTP server na portu definovaném argumentem příkazového řádku. Při tom umožňuje zpracovat libovolný počet požadavků zadaných na standardní vstup. Po každém požadavku vytiskne na standardní výstup odpověď. Po uzavření standardního vstupu vyřídí právě zpracovávané požadavky z HTTP serveru a končí. Importuje `sec.py` jako modul a lze říci, že je multi-klientem SEC démonu, protože každý dotaz z HTTP serveru má svého klienta k SEC démonu.

SEC démon vyžaduje ke svému běhu úpravu znalostní báze pro nástroje pro sdílení znalostní báze, jejich kompilaci a kompilaci nástroje Figa. Vše je zahrnuto do `Makefile` komponenty pro sémantické obohacení. Po spuštění se SEC démon pokusí připojit na sdílenou znalostní bázi pomocí již zmíněného modulu (v kapitole 6.1), popř. spustí démon pro sdílení znalostní báze a připojí si ji.

Nástroje NER a Figa jsem importoval do SEC démona jako moduly. Abych zabezpečil běh SEC démona, nastavil jsem odchytávání signálu `SIGSEGV` a vymezil čas na zpracování (timeout) u funkcí, kde bylo nebezpečí zacyklení. Volání funkce s vymezeným časem jsem implementoval pomocí podprocesu. Pro různé nástroje NER jsem vytvořil třídu `NERTemplate` dle návrhu v kapitole 5.2 a Petriho sítě (tamtéž na obrázku 5.2), jenž byla výborným vodítkem při implementaci. Nemohu ale s jistou tvrdit, že jsem implementoval přesně tuto Petriho síť, protože se mi nepodařilo zachovat přesně stejnou úroveň abstrakce. Do třídy `NERTemplate` jsem implementoval syntaktický analyzátor dle specifikace výstupu v kapitole 5.2.4. Obalení stávajícího nástroje NER tak vyžaduje pouze implementaci třídy, jenž je potomkem třídy `NERTemplate`, a redefinici metody `_process`, popř. i metod `_start` a `_stop` (analogie ze zmíněného návrhu Petriho sítí), a aby metoda `_process` vracela zpracovaný i prostý výstup. Tedy aby se obalovaný nástroj NER řídil dle zmíněné specifikace.

Rozdíl oproti návrhu komponenty pro sémantické obohacení je v tom, že jazyk Python nepodporuje paralelní běh vláken, a z toho důvodu jsem je nahradil podprocesy. Dalším důvodem bylo jednodušší zotavení SEC démonu z chyb, jenž z počátku nástroje NER a Figa vykazovaly. Chyba při zpracování jednoho požadavku by neměla ovlivnit ostatní požadavky.

6.3 Úpravy existujících nástrojů

Dle návrhu vylepšení nástroje NER (kapitola 5.3) jsem pro nástroj NER implementoval modul pro hledání dat („datumů“) ve vstupním textu. Regulární výrazy jsem vytvořil dle různých anglických formátů data. Pro převod do formátu ISO 8601 (formát rok-měsíc-den) jsem využil modul *dateutil*. Implementovaný modul pro hledání dat má problém rozpoznat rozdíl mezi britským (den/měsíc/rok) a americkým (měsíc/den/rok) formátem[1], pokud první i druhé číslo je menší než třináct. Jako výchozí je preferován americký formát. Např. datum „1/2/2010“ – je-li britské, je ekvivalentní k datu „2010-02-01“, je-li americké, pak k datu „2010-01-02“. Dále může nastat problém, je-li rok zadaný pouze dvojčíslím (např. „Jul 18 ’20“). V takovém případě není jisté, zdali se jedná o rok 1920 či 2020. Toto řeší využitý modul *dateutil* dle toho, který rok je bližší aktuálnímu roku (zde je to rok 2020).

Dále jsem nástroj NER propojil se sdílenou znalostní bází pomocí modulu zmíněného v sekci 6.1. Propojení jsem provedl tak, aby přístup k nástroji NER mohl být pořád stejný. Pokud se tedy připojení napoprvé po spuštění nástroje NER nezdaří, je z něj spuštěn démon pro sdílení znalostní báze a proběhne druhý pokus o připojení. Také jsem nástroj NER rozšířil o serializaci předzpracovaných dat ze znalostní báze, aby nemusela být při každém

[1] viz https://www.englishclub.com/vocabulary/time-date.htm

startu předzpracována znovu. Propojil jsem jej s hlavičkou znalostní báze, aby po změně znalostní báze (přidání atributu nebo změně pořadí atributů) nemusel být upraven.

Ve znalostní bázi byl typ `person` a typ `artist`. Při zavádění podtypů bylo vyžadováno, aby typ `artist` byl změněn na podtyp obecnějšího typu `person`. Tuto změnu jsem nejprve provedl v hlavičce znalostní báze a následně v nástroji pro tvorbu znalostní báze, generátoru vstupu pro tvorbu datových souborů Figa (propojení s hlavičkou znalostní báze provedl Ing. Otrusina) a nakonec v nástroji NER.

Kapitola 7

Testování komponenty pro sémantické obohacení a jejích podčástí

V této kapitole je popsáno, jakými experimenty a testy byla prokázána funkčnost implementovaných nástrojů a modulů.

Nejprve je popsán postup testování démona pro sdílení znalostní báze a měření jeho rychlosti. Následuje popis testování SEC s využitím vytvořených testovacích skriptů a na závěr jsou uvedeny informace o praktickém nasazení SEC v jiných projektech Výzkumné skupiny znalostních technologií (KNOT), v rámci kterých byla dále ověřena funkčnost poskytovaných služeb.

7.1 Testování démona pro sdílení znalostní báze

Démon pro sdílení znalostní báze a knihovnu jsem po implementaci otestoval na serveru `minerva3.fit.vutbr.cz` s operačním systém GNU/Linux (Ubuntu server 16.04.3 LTS), procesorem Intel(R) Xeon(R) CPU E5-2620 v3 na 2.40 GHz, diskovým polem o kapacitě 22 TB (6 disků o kapacitě 6 TB v HW RAID 6) a 62 GiB RAM.

Nejprve jsem testoval funkci na redukované znalostní bázi s deseti entitami o velikosti 3,4 KiB. Ke kontrole práce s pamětí jsem využil program `valgrind`. Pomocí něj jsem ověřil, že démon pro sdílení znalostní báze a ani knihovna nemá žádné úniky paměti.

Dále jsem měřil rychlost a náročnost na paměť na současné znalostní bázi obsahující přes 5 milionů entit s velikostí přesahující 2 GiB. K měření rychlosti jsem využil cíl `speedtest` souboru `Makefile`. Změřil jsem, že načtení takto veliké znalostní báze trvá přibližně 51 s ($\sim$ 41,745 MiB/s) a že předzpracovaná znalostní báze je přibližně o 20 % větší než původní (nárůst z 2129 MiB na 2563 MiB). Načtení předzpracované znalostní báze však proběhne do 2 s (dle vyrovnávací paměti).

Demonstrační program `benchmark.py` byl mimo jiné napsán i pro účel změření rychlosti přístupu k libovolnému sloupci/řádku v předzpracované znalostní bázi. S jeho využitím jsem změřil a vypočítal průměrnou rychlost přístupu k libovolnému sloupci/řádku. Z deseti měření skrz celou znalostní bázi (5 490 110 řádků) jsem vypočítal průměr ($\sim$ 8,168 s) a z něj vypočítal průměrnou rychlost přístupu k libovolnému sloupci/řádku znalostní báze $\sim$ 1,5 μs.

7.2 Testování komponenty pro sémantické obohacení na serverech KNOT a superpočítači Salomon

Pro otestování funkce komponenty pro sémantické obohacení textu jsem napsal několik cílů do souboru `Makefile`. Těmito testy jsem ověřil funkci všech služeb. Pomocí souboru `Makefile` jsem umožnil testovaní hlavních služeb po každé změně znalostní báze, nástroje Figa nebo nástroje NER. Mezi ně patří služba `annotate` pro výstupní formáty HTML, XML, SXML, TEXT, INDEX 2, RDF a NIF, dále služby `get_raw_annotations`, `get_entities`, `get_entity_by_uri`, `get_entity_types_and_attributes` a `get_kb_version`. Testovací sada služby `annotate` obsahuje třináct vstupních souborů. Výstup jsem zkontroloval manuálně. Všechny testy proběhnou po volání příkazu `make test_sec`, popř. příkazu `make test_net`. Vždy se uloží výstupy a čas uplynulý na jejich zpracování. Příkazem `make test_diff_prepare_ref` se aktuální výstupy a naměřený čas určí jako referenční pro porovnání s výstupy budoucího testování.

Do souboru `Makefile` jsem napsal cíle pro testování různých konfigurací a vstupních textů.
Součásti testů pro servery KNOT:

- `test_sec.py` jednoduchý skript pro otestování funkce HTTP serveru,
- `test/text/` texty extrahované z Wikipedie,
- `test/gen_query.py` skript pro generování příkazů pro SEC,
- `test/print_annotation.py` jednoduchý skript pro extrakci anotace z výsledku,
- `sge/sec.sh` skript pro využití v systému dávkového zpracování úloh SGE[1],
- `Makefile` obsahuje cíle pro testování různých konfigurací a vstupních textů.

Testování na superpočítači Salomon:

- využití `sge/sec.sh` pro integraci do řetězce pro zpracování korpusových dat[2],
- vytvořil jsem skripty `queue_client.py` a `queue_server.py`
  - zajišťují vytvoření vlastní očíslované fronty úloh pro zpětnou vazbu při zpracování (aby bylo zřejmé, které úlohy již byly zpracovány a které nikoli),
  - umožňují vyčkání na předzpracování znalostní báze nástroje SEC.

7.3 Nasazení komponenty pro sémantické obohacení v praxi

Vzhledem k tomu, že jsem na diplomové práci začal pracovat už v průběhu bakalářského studia, bylo vytvořené řešení v různých fázích vývoje prakticky nasazeno v řadě projektů. Následuje přehled těch nejvýznamnějších.

Komponenta pro sémantické obohacení textu se stala součástí nástroje 4A vytvořeného v rámci disertační práce doktora Dytrycha [31] a projektu Decipher (popsaného v sekci 3.3).

[1] SGE — http://www.fit.vutbr.cz/CVT/cluster/sge.php
[2] http://knot.fit.vutbr.cz/corpproc/corpproc_cs.html

V architektuře tohoto nástroje je z historických důvodů označována jako „SEC API server“[3] (v projektu Decipher byl pojmem SEC označen server složený ze 2 komponent, kde první byla SEC API, později přejmenovaná na SEC, a druhá SEC Store API pro indexaci a vyhledávání dokumentů, jejíž vývoj byl později ukončen). Ve všech experimentech popisovaných v dané disertační práci (kromě úvodního), byla využita komponenta pro sémantické obohacení. Moduly anotačního serveru využívají služeb SEC přes protokol HTTP (byť je zde implementována i možnost spuštění jako podprocesu, se kterým se komunikuje přes standardní vstup a výstup) a jako výstupní formát se využívá SXML. Byla tak prakticky ověřena jak funkčnost služeb pro anotování textu nástrojem NER, získání entit ze znalostní báze apod., tak i funkčnost testovacího režimu.

Po úspěšném ukončení projektu Decipher došlo k zařazení SEC mezi hlavní produkty Výzkumné skupiny znalostních technologií (KNOT). Veřejně je HTTP server komponenty pro sémantické obohacení dostupný na adrese `http://sec.fit.vutbr.cz/` a portu `8082`, kde si jej mohou vyzkoušet potenciální zájemci o spolupráci se skupinou KNOT, ale i tvůrci konkurenčních systémů, kteří hledají nástroje pro srovnání.

Dávid Prexta se ve své bakalářské práci [25] zabývá tvorbou nástroje pro srovnávání různých nástrojů NER. SEC ve své práci využil pro získání jednotného rozhraní pro různé nástroje NER. Každý ze srovnávaných nástrojů je obalen modulem pro SEC a následně je pomocí SEC s jednotlivými moduly zpracován text referenční datové sady. Výstupy ve formátu NIF se porovnávají s referenčním výstupem a vyhodnocují se chyby, kterých se jednotlivé nástroje dopustily. Protože se výstupy všech nástrojů s využitím SEC sjednocují i na využití stejné znalostní báze, jsou výstupy srovnání snadno strojově zpracovatelné a umožňují mimo jiné i rozšiřování a opravy znalostní báze.

Štěpán Vrša ve své bakalářské práci [32] navrhl a implementoval systém pro aktualizaci anotací v korpusech rozsáhlých textových dat. V rámci svojí práce implementoval modul MG4J_API pro SEC démon s využitím třídy `NERTemplate`. Umožňuje tak využití rozhraní anotačního nástroje pro SEC ke čtení anotací z indexů rozsáhlých textových korpusů a následné jednoduché zpracování zpětné vazby uživatelů k těmto anotacím. Využití SEC umožnilo integraci anotačního nástroje s minimálním úsilím (rozsah změn je v jednotkách řádků kódu pro výběr daného nástroje NER ve volání příslušné služby SEC) a Štěpán Vrša se tak mohl hlouběji věnovat efektivní práci s daty anotovaného korpusu.

SEC se využívá i v dalších aktuálních projektech. Při zpracování velkých korpusových dat v projektu Corpora Processing Software[4] je anotování korpusu s využitím SEC jedním z finálních kroků zpracování. Většina bakalářských prací ve skupině KNOT, které se zabývají zpracováním a indexací korpusových dat či dotazováním nad těmito indexy (např. [33]), tak přímo či nepřímo využívá SEC. Pomocí SEC tak již byly zpracovány stovky TB textových dat z několika verzí korpusu Common Crawl[5], z anglické Wikipedie[6] apod.

[3] `http://knot.fit.vutbr.cz/annotations/directions_for_use_annot_server_en.html`
[4] `http://knot.fit.vutbr.cz/corpproc/`
[5] `http://commoncrawl.org/`
[6] `https://dumps.wikimedia.org/`

Kapitola 8

Závěr

V této diplomové práci byla navrhnuta a implementována komponenta pro sémantické obohacení textu (SEC), slučující dostupné komponenty pro rozpoznávání pojmenovaných entit v textu a další komponenty s podobným zaměřením pod jedno rozhraní.

Byl analyzován a popsán nástroj pro rozpoznání pojmenovaných entit v textu (NER) Výzkumné skupiny znalostních technologií FIT VUT v Brně, nástroj pro vyhledávání entit v textu (Figa) a znalostní báze obsahující entity, jejich typy a atributy. Pro komponenty NER a znalostní bázi byly navrženy specifikace zjednodušující implementaci navržených nástrojů. Specifikace výstupu nástrojů NER sjednocuje výstup různých nástrojů NER. Byla navržena hlavička znalostní báze popisující všechny typy, podtypy a jejich atributy vyskytující se ve znalostní bázi. Specifikace znalostní báze byla vytvořena pro navrženou hlavičku znalostní báze. Umožňuje již existujícím nástrojům z hlavičky znalostní báze čerpat informace o znalostní bázi a jimi nahradit informace uvedené jako konstanty přímo ve zdrojovém kódu těchto nástrojů.

Pro úsporu operační paměti i času při zpracování znalostní báze byl navržen nástroj „Démon pro znalostní bázi“, jehož úkolem je znalostní bázi předzpracovat a úsporně uložit do sdílené paměti. Dále zrychlit přístup k jednotlivým řádkům, popř. i sloupcům znalostní báze a umožnit šetřit paměť předáním ukazatele na požadovaný řetězec. Byly popsány a dále rozebrány dva navržené způsoby implementace démona pro znalostní bázi. Z těchto navržených způsobů bylo vybráno tak, aby ke zvýšení rychlosti nedocházelo za cenu znatelného navýšení spotřeby paměti.

Vzhledem k pomalé inicializaci nástroje NER byly navrženy a implementovány dvě úpravy tohoto nástroje. První úprava spočívá ve využití sdílené znalostní báze, což ušetří jak čas, tak operační paměť. Další úprava se týká předzpracování dat ze znalostní báze nástrojem NER. Předzpracovaná data se vzhledem ke znalostní bázi nemění, což je umožňuje serializovat do souboru, načítat je z něj při další inicializaci a pouze kontrolovat, zda se znalostní báze změnila či nikoli. V případě změny je pak nutno data ze znalostní báze znovu předzpracovat a serializovat.

S využitím nástrojů pro sdílení znalostní báze a nástroje NER byla navržena a implementována komponenta pro sémantické obohacení textu (SEC), která byla následně otestována a začleněna do celé řady projektů. V roce 2015 se stala oficiálním produktem Výzkumné skupiny znalostních technologií FIT VUT v Brně a ve stejném roce následně i součástí druhého produktu s názvem *Programy pro zpracování korpusů*.

Do budoucna by komponentou pro sémantické obohacení textu bylo možné rozšířit o posílání otevřeného popisovače souborů na vstupní text, což by mohlo snížit režii v komunikaci mezi klientem a SEC démonem. Jelikož je ve vývoji nástroj NER pro český jazyk,

bude možné jej importovat a vyzkoušet jako nový modul komponenty pro sémantické obohacení textu. Dále by bylo možné rozšířit modul pro hledání dat („datumů“) ve vstupním textu a pokusit se vylepšit rozpoznání mezi britským a americkým formátem data, mezi nimiž je zaměněn den a měsíc. Narazíme-li v textu na datum, kde první nebo druhé číslo je větší než dvanáct, pak je jednoduché dodatečně zaměnit den a měsíc za předpokladu, že si nejistá data ukládáme do seznamu.

Literatura

[1] SMRŽ, Pavel; OTRUSINA, Lubomír; KOUŘIL, Jan; DYTRYCH, Jaroslav. *DECIPHER Semantic Annotator* [online]. 2013 [cit. 2018-01-16]. Dostupné z: http://cordis.europa.eu/docs/projects/cnect/1/270001/080/deliverables/001-DecipherD431SemanticAnnotatorv01.pdf. Technická zpráva.

[2] IT4INNOVATIONS. *Salomon Cluster: Hardware Overview* [online]. 2017 [cit. 2018-01-16]. Dostupné z: https://docs.it4i.cz/salomon/hardware-overview/.

[3] IT4INNOVATIONS. *Salomon Cluster: Job Submission and Execution* [online]. 2017 [cit. 2018-01-16]. Dostupné z: https://docs.it4i.cz/salomon/job-submission-and-execution/.

[4] OTRUSINA, Lubomír. *Knowledge Base (iotrusina)* [online]. 2017 [cit. 2018-01-16]. Dostupné z: http://knot.fit.vutbr.cz/Decipher_NER/Decipher_NER-cs.html#Knowledge_Base_.28iotrusina.29.

[5] OTRUSINA, Lubomír. *Tvorba automatů (iotrusina)* [online]. 2017 [cit. 2018-01-16]. Dostupné z: http://knot.fit.vutbr.cz/Decipher_NER/Decipher_NER-cs.html#Tvorba_automat.C5.AF_.28iotrusina.29.

[6] HAVRAN, Jan. *Integrace projektu Ner4 (xhavra13)* [online]. 2017 [cit. 2018-01-16]. Dostupné z: http://knot.fit.vutbr.cz/Decipher_NER/Decipher_NER-cs.html#Integrace_projektu_Ner4_.28xhavra13.29.

[7] CUDRÁK, Miloš. *Změny dokumentu v editoru anotací* [online]. Brno, CZ, 2014 [cit. 2018-01-10]. Dostupné z: http://www.fit.vutbr.cz/study/DP/DP.php?id=15629. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroslav DYTRYCH.

[8] MAGDOLEN, Matej. *Jak funguje ner.py? (xmagdo00)* [online]. 2017 [cit. 2018-01-16]. Dostupné z: http://knot.fit.vutbr.cz/Decipher_NER/Decipher_NER-cs.html#Jak_funguje_ner.py.3F_.28xmagdo00.29.

[9] OTRUSINA, Lubomír; MAGDOLEN, Matej. *Nástroj ner.py (xmagdo00, iotrusina)* [online]. 2017 [cit. 2018-01-16]. Dostupné z: http://knot.fit.vutbr.cz/Decipher_NER/Decipher_NER-cs.html#N.C3.A1stroj_ner.py_.28xmagdo00.2C_iotrusina.29.

[10] KERRISK, Michael. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook* [online]. No Starch Press, 2010 [cit. 2018-01-13]. No Starch Press Series. ISBN 978-1-59327-220-3. Dostupné z: https://books.google.cz/books?id=2SAQAQAAQBAJ.

[11] KERRISK, Michael. *shm_overview(7) – Linux manual page* [online]. 2017 [cit. 2018-01-13]. Dostupné z: http://man7.org/linux/man-pages/man7/shm_overview.7.html.

[12] PYTHON SOFTWARE FOUNDATION. *General Python FAQ* [online]. 2018 [cit. 2018-01-16]. Dostupné z: https://docs.python.org/3/faq/general.html.

[13] PYTHON SOFTWARE FOUNDATION. *History and License* [online]. 2018 [cit. 2018-01-16]. Dostupné z: https://docs.python.org/3/license.html.

[14] PYTHON SOFTWARE FOUNDATION. *Python Documentation by Version* [online]. 2018 [cit. 2018-01-16]. Dostupné z: https://www.python.org/doc/versions/.

[15] PETERSON, Benjamin. *PEP 373 – Python 2.7 Release Schedule* [online]. 2016 [cit. 2018-01-16]. Dostupné z: http://legacy.python.org/dev/peps/pep-0373/.

[16] DOWNEY, A.; ELKNER, J.; MEYERS, C. *How to Think Like a Computer Scientist: Learning with Python*. Samurai Media Limited, 2016. ISBN 9789888406784. Dostupné také z: https://books.google.cz/books?id=m-AavgAACAAJ.

[17] FREE SOFTWARE FOUNDATION. *Bash Reference Manual* [online]. 2016 [cit. 2018-01-16]. Dostupné z: https://www.gnu.org/software/bash/manual/bash.html.

[18] FREE SOFTWARE FOUNDATION. *Bash Versions* [online]. 2017 [cit. 2018-01-16]. Dostupné z: https://ftp.gnu.org/gnu/bash/.

[19] SIEVER, Ellen. *LINUX v kostce*. Praha: Computer Press, 1999. ISBN 80-7226-227-0.

[20] FIELDING, Roy T.; GETTYS, James; MOGUL, Jeffrey C.; NIELSEN, Henrik Frystyk; MASINTER, Larry; LEACH, Paul J.; BERNERS-LEE, Tim. *Hypertext Transfer Protocol – HTTP/1.1* [online]. Internet Engineering Task Force (IETF), 1999 [cit. 2018-05-17]. Dostupné z: https://tools.ietf.org/html/rfc2616. Technická zpráva.

[21] BELSHE, Mike; THOMSON, Martin; PEON, Roberto. *Hypertext Transfer Protocol Version 2 (HTTP/2)* [online]. Internet Engineering Task Force (IETF), 2015 [cit. 2018-05-17]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc7540. Technická zpráva.

[22] BRAY, Tim (ed.). *The JavaScript Object Notation (JSON) Data Interchange Format* [online]. Internet Engineering Task Force (IETF), 2017 [cit. 2018-05-17]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc8259. Technická zpráva.

[23] ŠVAŇA, Miloš. *Čištění, extrakce textu a převod webových stránek do vertikálního formátu* [online]. Brno, CZ, 2016 [cit. 2018-01-16]. Dostupné z: http://www.fit.vutbr.cz/study/DP/BP.php?id=18729. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroslav DYTRYCH.

[24] ŠVAŇA, Miloš. *Vertikál* [online]. 2016 [cit. 2018-01-16]. Dostupné z: http://knot.fit.vutbr.cz/corpproc/vertical_cs.html.

[25] PREXTA, Dávid. *Porovnávání anotačních nástrojů* [online]. Brno, CZ, 2017 [cit. 2018-01-16]. Dostupné z: http://www.fit.vutbr.cz/study/DP/BP.php?id=20061. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroslav DYTRYCH.

[26] WORLD WIDE WEB CONSORTIUM. *HTML 5.2* [online]. 2017 [cit. 2018-05-05]. Dostupné z: https://www.w3.org/TR/html/.

[27] PÍSEK, Slavoj. *HTML: začínáme programovat*. 4., aktualiz. vyd. Praha: Grada, 2014. ISBN 978-80-247-5059-0.

[28] WORLD WIDE WEB CONSORTIUM. *RDF 1.1 Primer* [online]. 2014 [cit. 2018-05-12]. Dostupné z: https://www.w3.org/TR/rdf11-primer/.

[29] WORLD WIDE WEB CONSORTIUM. *Extensible Markup Language (XML) 1.1* [online]. 2006 [cit. 2018-05-12]. Dostupné z: https://www.w3.org/TR/xml11/.

[30] BRADLEY, Neil. *XML: kompletní průvodce*. Praha: Grada, 2000. ISBN 80-7169-949-7.

[31] DYTRYCH, Jaroslav. *Sémantická anotace textu* [online]. Brno, CZ, 2017 [cit. 2018-01-10]. Dostupné z: http://www.fit.vutbr.cz/study/DP/PD.php?id=548. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Pavel SMRŽ.

[32] VRŠA, Štěpán. *Systém pro aktualizaci anotací v korpusech* [online]. Brno, CZ, 2017 [cit. 2018-01-16]. Dostupné z: http://www.fit.vutbr.cz/study/DP/BP.php?id=20153. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroslav DYTRYCH.

[33] PANOV, Sergey. *Indexování a prohledávání sémanticky anotovaných textů* [online]. Brno, CZ, 2017 [cit. 2018-01-16]. Dostupné z: http://www.fit.vutbr.cz/study/DP/BP.php?id=20056. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Pavel SMRŽ.